Project number IST-25582

CGL

Computational Geometric Learning

Lines Through Segments in 3D Space

STREP

Information Society Technologies

Period covered:	November 1, 2011–October 31, 2012
Date of preparation:	October 21, 2012
Date of revision:	October 21, 2012
Start date of project:	November 1, 2010
Duration:	3 years
Project coordinator name:	Joachim Giesen (FSU)
Project coordinator organisation:	Friedrich-Schiller-Universität Jena Jena, Germany

Lines Through Segments in 3D Space*

Efi Fogel, Michael Hemmer, Asaf Porat, and Dan Halperin

The Blavatnik School of Computer Science, Tel Aviv University
{efifogel,mhsaar,asafpor}@gmail.com, and danha@post.tau.ac.il

Abstract. Given a set $\mathcal{S}$ of n line segments in three-dimensional space, finding all the lines that simultaneously intersect at least four line segments in $\mathcal{S}$ is a fundamental problem that arises in a variety of domains. We refer to this problem as *the lines-through-segments problem*, or LTS for short. We present an efficient output-sensitive algorithm and its implementation to solve the LTS problem. The implementation is exact and properly handles all degenerate cases. To the best of our knowledge, this is the first implementation for the LTS problem that is (i) output sensitive and (ii) handles all degenerate cases. The algorithm runs in $O((n^3 + I) \log n)$ time, where I is the output size, and requires $O(n \log n + J)$ working space, where J is the maximum number of output elements that intersect two fixed line segments; I and J are bounded by $O(n^4)$ and $O(n^2)$, respectively. We use CGAL arrangements and in particular its support for two-dimensional arrangements in the plane and on the sphere in our implementation. The efficiency of our implementation stems in part from careful crafting of the algebraic tools needed in the computation. We also report on the performance of our algorithm and its implementation compared to others. The source code of the LTS program as well as the input examples for the experiments can be obtained from http://acg.cs.tau.ac.il/projects/lts.

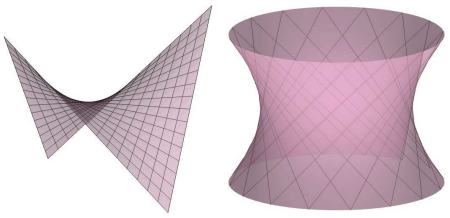
1 Introduction

Given a set $\mathcal{S}$ of line segments in $\mathbb{R}^3$, we study the *lines-through-segments* (LTS) problem, namely, the problem of computing all lines that simultaneously intersect at least four line segments in $\mathcal{S}$. LTS is a fundamental problem that arises in a variety of domains. For instance, solving the LTS problem can be used as the first step towards solving the more general problem of finding all lines tangent to at least four geometric objects taken from a set of geometric objects. The latter is ubiquitous in many fields of computation such as computer graphics (visibility computations), computational geometry (line transversal), robotics and automation (assembly planning), and computer vision. Computing visibility information, for example, is crucial to many problems in computer graphics,

* This work has been supported in part by the 7th Framework Programme for Research of the European Commission, under FET-Open grant number 255827 (CGL—Computational Geometry Learning), by the Israel Science Foundation (grant no. 1102/11), by the German-Israeli Foundation (grant no. 969/07), and by the Hermann Minkowski–Minerva Center for Geometry at Tel Aviv University.

vision, and robotics, such as computing umbra and penumbra cast by a light source [7].

The number of lines that intersect four lines in $\mathbb{R}^3$ is 0, 1, 2, or infinite. Brönnimann et al. [6] showed that the number of lines that intersect four arbitrary line segments in $\mathbb{R}^3$ is 0, 1, 2, 3, 4, or infinite. They also showed that the lines lie in at most four maximal *connected components*.[1]



(a) A hyperbolic paraboloid. **(b)** A hyperboloid of one sheet.

Fig. 1: Two configurations of lines segments in which an infinite number of lines intersect four line segments.

A straightforward method to find all the lines that intersect four lines, given a set of n lines, examines each quadruplet of lines using, for example, the Plücker coordinate representation. This method has been used by Hohmeyer and Teller [17] and also described by Redburn [15]. It was later used by Everett et al. [10] as a building block for the problem of finding line transversals (the set of lines that intersect all given line segments). The running time of this method is $O(n^4)$.

The combinatorial complexity of the lines that intersect four line segments of a set of n line segments is $\Theta(n^4)$ (counting maximal connected components). The lower bound can be established by placing two grids of $n/2$ line segments each in two parallel planes and passing a line through every two intersection points, one from each grid. However, in many cases the number of output lines is considerably smaller. The size of the output tends to be even smaller, when the input consists of line segments (as opposed to lines), which is typically the case in practical problems, and it is expected to decrease with the decrease of the input line-segment lengths.

We present an efficient output-sensitive algorithm, and its complete and robust implementation, that solves the LTS problem in three-dimensional Euclidean space.[2] The implementation is complete in the sense that it handles all degenerate cases and guarantees exact results. Examples of degenerate cases are: A line segment may degenerate to a point, several segments may intersect, be coplanar, parallel, concurrent, lie on the same supporting line, or even overlap. To the best of our knowledge, this is the first implementation for the LTS problem that is (i) output sensitive and (ii) handles all degenerate cases. The algorithm utilizes the idea of McKenna and O'Rouke [13] to represent the set of lines that intersect three lines as a rectangular hyperbola with a vertical and a horizontal asymptotes in $\mathbb{R}^2$. However, as opposed to their algorithm, which takes $O(n^4\alpha(n))$ time, our algorithm is output sensitive and its asymptotic time and space complexities are $O((n^3 + I)\log n)$ and $O(n \log n + J)$, respectively,

[1] Two lines tangent to the same four line segments are in the same connected component iff one of the lines can be continuously moved into the other while remaining tangent to the same four line-segments.

[2] A short version of the extended abstract was presented in EuroCG this year.

where n is the input size, I is the output size, and J is the maximum number of output elements that intersect two fixed line segments; I and J are bounded by $O(n^4)$ and $O(n^2)$, respectively. The algorithm can be trivially altered to accept a constant $c \geq 4$ and compute all lines that simultaneously intersect exactly, or at least, c line segments from the input set. In addition, the algorithm can easily be changed to compute transversals to line segments in $\mathbb{R}^3$ [6].

A related problem to the problem at hand is the *the lines-tangent-to-polytopes problem*, or LTP for short. Formally, given a set $\mathcal{P}$ of n convex polytopes in three-dimensional space, the objective is to find all the lines that are simultaneously tangent to quadruples of polytopes in $\mathcal{P}$. This, in turn, can be generalized to the problem of determining the visibility between objects. In many cases a solution to the visibility or LTP problems can also serve as a solution to the LTS problem. Brönnimann et al. [5] provide a non-output sensitive solution to the visibility problem. It runs in time $O(n^2k^2 \log n)$ for a scene of k polyhedra of total complexity n (although their algorithm is sensitive to the size of the 2D visibility skeletons, calculated during the process). Devillers et al. [8] introduce efficient algebraic methods to evaluate the geometric predicates required during the visibility computation process.

Our algorithms are implemented on top of the Computational Geometry Algorithm Library (CGAL) [18]. The implementation is mainly based on the *2D Arrangements* package of the library [20]. This package supports the robust and efficient construction and maintenance of arrangements induced by curves embedded on certain orientable two-dimensional parametric surfaces in three-dimensional space [2, 19], and robust operations on them.[3] The implementation uses in particular 2D arrangements of rectangular hyperbolas with vertical and horizontal asymptotes in the plane and 2D arrangements of geodesic arcs on the sphere [1].

The rest of this paper is organized as follows. In Section 2 we introduce the necessary terms and definitions and the theoretical foundation of the algorithm that solves the LTS problem. In Section 3 we present a limited version of the algorithm. In Section 4 we describe the specially tuned algebraic tools used in the implementation. We report on experimental results in Section 5 and suggest future directions in Section 6. Because of space limitation many details of the analysis (Section 2) and the algorithm (Section 3) are omitted.

2 Representation

This section discusses the encoding of all lines that intersect two fixed line segments, S_1 and S_2, and a third line segment, S_3. We represent a line $L \subset \mathbb{R}^3$ by a point $p \in \mathbb{R}^3$ and a direction $d \in \mathbb{R}^3 \setminus \{\mathcal{O}\}$ as $L(t) = p + t \cdot d$, where $\mathcal{O}$ denotes the origin and $t \in \mathbb{R}$. Clearly, this representation is not unique. A segment $S \subset L \subset \mathbb{R}^3$ is represented by restricting t to the interval $[a, b] \subset \mathbb{R}$. We refer to $S(a)$ and $S(b)$ as the source and target points, respectively, and set

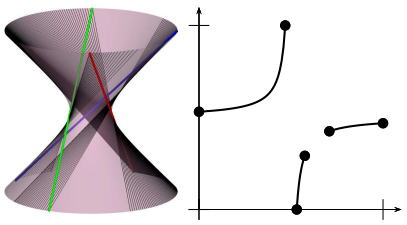
[3] Arrangements on surfaces are supported as of CGAL version 3.4, albeit not documented yet.

$a = 0$ and $b = 1$. We denote the underlying line of a line segment S by $L(S)$. Two lines are *skew* if they are not coplanar. Three or more lines are *concurrent* if they all intersect at a common point.

2.1 S_1 and S_2 Are Skew

Given two lines L_1 and L_2 we define a map $\Psi_{L_1L_2}$, which maps a point in $\mathbb{R}^3$ to a set in $\mathbb{R}^2$, as follows: $\Psi_{L_1L_2}(q) = \{(t_1, t_2) \in \mathbb{R}^2 \,|\, L_1(t_1),\ L_2(t_2),\ \text{and } q \text{ are collinear}\}$. The set $\Psi_{L_1L_2}(q)$, which might be empty, corresponds to all lines that contain q and intersect L_1 and L_2. Now, consider the pair $(t_1, t_2) \in \mathbb{R}^2$. If $L_1(t_1) \neq L_2(t_2)$, then this pair uniquely defines the line that intersects L_1 and L_2 at $L_1(t_1)$ and $L_2(t_2)$, respectively. Thus, for skew lines L_1 and L_2 there is a canonical bijective map between $\mathbb{R}^2$ and all lines that intersect L_1 and L_2. It follows that for disjoint lines L_1 and L_2 and a third line L_3 the set $\Psi_{L_1L_2}(L_3)$ is sufficient to represent all lines that intersect L_1, L_2, and L_3, where $\Psi_{L_1L_2}(L_3) = \{\Psi_{L_1L_2}(q) \,|\, q \in L_3\}$. Similarly, we define $\Psi_{s_1s_2}(q) = \{(t_1, t_2) \,|\, S_1(t_1),\ S_2(t_2),\ \text{and } q \text{ are collinear}, (t_1, t_2) \in [0, 1]^2\}$ for two line segments S_1 and S_2. The characterization of $\Psi_{S_1S_2}(S_3) = \{\Psi_{S_1S_2}(q) \,|\, q \in S_3\}$ serves as the theoretical foundation of the algorithm that solves the LTS problem presented in Section 3. As $\Psi_{S_1S_2}(x) = \Psi_{L(S_1)L(S_2)}(x) \cap [0, 1]^2$, it is sufficient to analyze $\Psi_{L_1L_2}(S_3)$ for a line segment S_3. We omit the complete characterization due to limited space, and only mention the generic case where S_1, S_2, and S_3 are pairwise skew below. The characterization is summerized by the following corollary.

Corollary 1. *$\Psi_{S_1S_2}(S_3) \subset \mathbb{R}^2$ is either a point, a one dimensional set consisting of line segments or arcs of a rectangular hyperbola with a vertical and a horizontal asymptotes, or a two-dimensional set bounded by linear segments or arcs of such hyperbolas.*



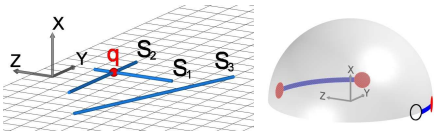
Consider the case in which the direction vectors of the underlying lines of the segments S_1, S_2, and S_3 are pairwise skew. From the complete analysis omitted here it follows that $\Psi_{L(S_1)L(S_2)}(L(S_3))$ is a rectangular hyperbola with a vertical and a horizontal asymptotes, and $\Psi_{S_1S_2}(S_3)$ consists of at most three maximal connected components, where each component represents a patch of a ruled surface. The figure depicted above shows three surface patches the lines of which intersect three skew line segments, S_1, S_2, and S_3, in $\mathbb{R}^3$. These surface patches are contained in a hyperboloid of one sheet; see Figure 1b. Also depicted is the point set $\Psi_{S_1S_2}(S_3)$.

2.2 S_1 and S_2 Intersect

Assume L_1 and L_2 intersect, and let $q = L_1(\tilde{t_1}) = L_2(\tilde{t_2})$ be the intersection point. The point $(\tilde{t_1}, \tilde{t_2})$ represents all lines that contain q. We represent these

lines by points on a semi open upper hemisphere centered at q. We define the additional map $\Xi_q : \mathbb{R}^3 \setminus \{q\} \to \mathbb{H}^2$ and $\Xi_q(p) \longmapsto d = s(p-q)/|p-q|$, with $s \in \{\pm 1\}$, such that $d \in \mathbb{H}^2 = \{p \,|\, p \in \mathbb{S}^2$ and p is lexicographically larger than $\mathcal{O}\}$.[4]

In the generic case a segment S maps to one or two geodesic arcs on $\mathbb{H}^2$. If S_3 is a point, or $L(S_3)$ contains q and S_3 does not, $\Xi_q(S)$ consists of a single point. If $q \in S_3$, we define $\Xi_q(S_3) = \mathbb{H}^2$. The left image of the figure above depicts three line segments, S_1, S_2, and S_3, such that S_1 and S_2 intersect at q (and S_3 does not). The right image depicts the mapping $\Xi_q(S_3)$, where $\Xi_q(S_3) = \{\Xi_q(p) \,|\, p \in S_3\}$. It consists of two geodesic arcs on $\mathbb{H}^2$.



2.3 S_1 and S_2 Are Collinear

The case where S_1 and S_2 are collinear completes the list of possible cases. If S_1 and S_2 do not overlap, the only line that can possibly intersect S_1 and S_2 is the line containing S_1 and S_2. Otherwise, the number of degrees of freedom of all the lines that intersect S_1 and S_2 is three. In any case S_1 and S_2 are handled separately. The handling does not involve a mapping to a two-dimensional surface.

We are now ready to describe our algorithm for solving the LTS problem in its full generality. Further details related to the variant cases handled are available at `http://acg.cs.tau.ac.il/projects/lts`.

3 The Algorithm

The input is a set $\mathcal{S} = \{S_1, \ldots, S_n\}$ of n line segments in $\mathbb{R}^3$. The output is a set of at most $O(n^4)$ elements. Assuming that an input line imposes a single intersection constraint, each element is either (i) a (one-dimensional) line that abides by at least four intersection constraints imposed by the line segments in $\mathcal{S}$, (ii) a (two-dimensional) ruled surface patch in $\mathbb{R}^3$, such that all lines that lie on the surface abide by at least four such intersection constraints, (iii) the set of all lines that contain a given point, which is, for example, the intersection of at least four concurrent lines, or (iv) the set of all lines that intersect a given line segment, which is the intersection of at least four input line segments. By default, however, we assume that a sub-segment that is the intersection of several overlapping line segments imposes a single intersection constraint, and a point that is either the intersection of several concurrent line segments, or simply a degenerate line segment, imposes two intersection constraints. The line segments that impose the constraints of an output element are referred to as the *generating line segments* of that element. It is possible to couple the output lines with their generating line segments.

[4] $\mathbb{H}^2$ is the union of an open half sphere, an open half circle, and a point.

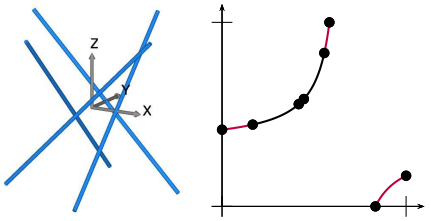
To simplify the exposition of the algorithm, we assume that the line segments are full-dimensional, may intersect pairwise only at discrete and distinct points, and no three line segments are coplanar. Due to limited space we only partially describe the algorithm that handles this case. The detailed description of the complete algorithm that handles all cases can be found at `http://acg.cs.tau.ac.il/projects/lts`. The complete algorithm also respects several different settings selected by the user. They are also listed in the appendix.

We transform the original three-dimensional LTS problem into a collection of two-dimensional problems and use two-dimensional arrangements to solve them, exploiting the plane-sweep algorithmic framework, which is output sensitive. We go over unordered pairs of lines segments in $\mathcal{S}$. For each pair, (S_i, S_j), we find all lines that intersect S_i, S_j, and two other line segments in $\mathcal{S}$, that have not been found yet in previous iterations; see Algorithm 1 for pseudo code.

Algorithm 1 Compute lines that intersect line segments $S_1, \ldots, S_n$.

```
1  for i = 1, ..., n − 3,
2    for j = n, ..., i + 3,
3      Construct arrangement A_{S_i S_j} induced by {Ψ_{S_i S_j}(S_k) | k = i + 1, ..., j − 1}.
4      Extract lines that intersect S_i and S_j from A_{S_i S_j}.
5      if S_i and S_j intersect,
6        Construct arrangement A^s_{S_i ∩ S_j} induced by {Ξ_{S_i ∩ S_j}(S_k) | k = i + 1, ..., j − 1}.
7        Extract lines that intersect S_i and S_j from A^s_{S_i ∩ S_j}.
```

In Line 3 of Algorithm 1 we construct the arrangement $\mathcal{A}_{S_iS_j}$ induced by the set of (possibly degenerate) hyperbolic arcs $\mathcal{C}_{ij} = \{\Psi_{S_iS_j}(S_k) \mid k = i+1, \ldots, j-1\}$. We process the line segments $S_{i+1}, \ldots, S_{j-1}$ one at a time to produce the inducing set $\mathcal{C}_{ij}$. Next, using a plane-sweep algorithm, we construct the arrangement $\mathcal{A}_{S_iS_j}$ induced by $\mathcal{C}_{ij}$. We store with each vertex and edge of the arrangement $\mathcal{A}_{S_iS_j}$ the sorted sequence of line segments that are mapped through $\Psi_{S_iS_j}$ to the points and curves that induce that cell. The segments are sorted by their indices. Observe that curves in $\mathcal{C}_{ij}$ may overlap; see Figure 2b.



(a) (b)

Fig. 2: (a) Four line segments, S_1, S_2, S_3, and S_4, supported by four lines of one ruling of a *hyperbolic paraboloid*, respectively; see also Figure 1a. (b) The arrangement $A_{S_1S_2}$. The edges drawn in purple are induced by two overlapping curves, one in $\Psi_{S_1S_2}(S_3)$ and the other in $\Psi_{S_1S_2}(S_4)$.

The generating line segments of every output element are immediately available from the sequences of line segments stored with vertices and edges. However, the role of these sequences extends beyond reporting. It turns out that some intersection points do not represent lines that intersect four line segments. An example of such a case occurs when either S_i or S_j intersects a third line segment, S_k. In such a case $\Psi_{S_iS_j}(S_k)$ consists of horizontal and vertical line seg-

ments. intersection point of the vertical and horizontal line segments does not represent a line that intersects four line segments and, thus, must be ignored. This case is detected by examining the sorted sequences of line segments.

Consider the case where S_i and S_j intersect at a point, say p. In this case we must output every line that contains p and abides by two additional intersection constraints. This information is not present in the planar arrangements constructed in Line 3. We can change the algorithm to construct an arrangement $\mathcal{A}_{S_iS_j}$ for every ordered pair (i, j) of indices of line segments in $\mathcal{S}$, where $\mathcal{A}_{S_iS_j}$ is induced by $\{\Psi_{S_iS_j}(S) \mid S \in \mathcal{S} \setminus \{S_i, S_j\}$, and filter out redundancies. While this modification does not increase the asymptotic complexity of the algorithm, our experiments show that constructing arrangements on the sphere instead exhibits much better performance in practice.

In Line 6 of Algorithm 1 we construct an arrangement on the sphere centered at the intersection point of S_i and S_j. The arrangement is induced by the set of (possibly degenerate) geodesic arcs $\mathcal{C}^s_{ij} = \{\Xi_{S_i\cap S_j}(S_k) \mid k = i+1, \ldots, j-1\}$. We process the line segments $S_{i+1}, \ldots, S_{j-1}$ one at a time to produce the inducing set $\mathcal{C}^s_{ij}$. When the underlying line of a line segment S_k contains the sphere center, $\Xi_{S_i\cap S_j}(S_k)$ consists of a single point. For each k, $i < k < j$, $\Xi_{S_i\cap S_j}(S_k)$ consists of either an isolated point or at most two geodesic arcs on the sphere; see Section 2.2. The pairwise intersections of the points and arcs in $\mathcal{C}^s_{ij}$ represent lines that intersect at least four input segments. Next, using a plane-sweep algorithm on the sphere, we construct the arrangement $\mathcal{A}^s_{S_i\cap S_j}$ induced by $\mathcal{C}^s_{ij}$. When $\Xi_{S_i\cap S_j}(S_k)$ consists of a single point it induces a single vertex in the arrangement. We store with each vertex and edge of the arrangement $\mathcal{A}^s_{S_i\cap S_j}$ the sorted sequence of line segments that are mapped through $\Xi_{S_i\cap S_j}$ to the points and geodesic arcs that induce that cell.

In Line 4 and Line 7 we extract the information and provide it to the user in a usable format. We refer to an arrangement cell that represents a valid output element as an eligible cell. We provide the user with the ability to iterate over eligible cells of different dimensions separately. This way, for example, a user can choose to obtain only the vertices that represent valid output lines. The eligibility of a given cell is immediately established from the sequence of line segments stored in that cell, and so are the generating line segments.

Constructing the arrangement $\mathcal{A}_{S_iS_j}$ and $\mathcal{A}^s_{S_i\cap S_j}$ are the dominant steps in the algorithm. For a given pair of indices (i, j), they are carried out in $O((n + k_{ij}) \log n)$ and $O((n + k^s_{ij}) \log n)$ time, respectively (using plane-sweep algorithms) where k_{ij} and k^s_{ij} are the numbers of intersections of the inducing hyperbolic and geodesic arcs, respectively. Thus, the entire process can be performed in $O((n^3 + I) \log n)$ running time, and $O(n + J)$ working space. Where n is the input size, I is the output size, and J is the maximum number of intersections in a single arrangement. I and J are bounded by $O(n^4)$ and $O(n^2)$, respectively.

4 Lazy Algebraic Tools

Our implementations are exact and complete. In order to achieve this, CGAL in general, and the CGAL *2D Arrangements* package [20] in particular, follows the *exact geometric-computation (EGC) paradigm* [22]. A naive attempt could realize this by carrying out each and every arithmetic operation using an expensive unlimited-precision number type. However, only the discrete decisions in an algorithm, namely the predicates, must be correct. This is a significant relaxation from the naive concept of numerical exactness. This way it is possible to use fast inexact arithmetic (e.g., double-precision floating-point arithmetic [9]), while analyzing the correctness. If the computation reaches a stage of uncertainty, the computation is redone using unlimited precision. In cases where such a state is never reached, expensive computation is avoided, while the result is still certified. In this context CGAL's Lazy kernel [14] is the state of the art, as it not only provides filtered predicates, but also delays the exact construction of coordinates and objects. While arithmetic is only carried out with (floating-point) interval arithmetic [4], each constructed object stores its construction history in a directed acyclic graph (DAG). Only in case the result of a predicate evaluated using interval arithmetic is uncertain, the DAG is evaluated using unlimited precision.

CGAL follows the *generic programming paradigm*; that is, algorithms are formulated and implemented such that they abstract away from the actual types, constructions, and predicates. Using the C++ programming language this is realized by means of class and function templates. CGAL's arrangement data structure is parameterized by a *traits* class that defines the set of curves types and the operations on curves of these types.

For the arrangement of geodesic arcs on the sphere we use the existing and efficient traits class that we have used in the past [1]. As this only requires a linear kernel, it uses CGAL's efficient Lazy Kernel [4]. However, in order to compute the planar arrangements of rectangular hyperbolic arcs with horizontal and vertical asymptotes, CGAL offered only a general traits class for rational functions, which was introduced in [16]. The class uses the general univariate algebraic kernel [3] of CGAL, which does not offer lazy constructions.

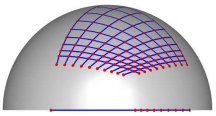
We modified the aforementioned traits class such that it is specialized for rectangular hyperbolic arcs with horizontal and vertical asymptotes. This way, it was possible to take advantage of the fact that in our case coordinates of intersection points are of algebraic degree at most 2. In particular, CGAL already offers a specific type[5] that represents such a number as $a+b\sqrt{c}$, where $a, b, c \in \mathbb{Q}$, This type can be used in conjunction with the lazy mechanism. Thus, the specialized traits constructs all point coordinates in a lazy fashion, which considerably speeds up the computation as shown in Section 5.

[5] CGAL::Sqrt_extension, part of the *Number Types* package of CGAL [12].

5 Experimental Results

We have conducted several experiments on three types of data sets. The first produces the worst-case combinatorial output and has many degeneracies. The second consists of transformed versions of the first and has many near-degeneracies. The third comprises random input. We report on the time consumption of our implementation, and compare it to those of other implementations. All experiments were performed on a Pentium PC clocked at 2.40 GHz.

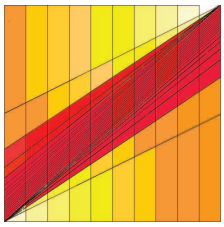
5.1 Grid



The Grid data set comprises 40 line segments arranged in two grids of 20 lines segments each lying in two planes parallel to the yz-plane; see Section 1. Each grid consists of ten vertical and ten horizontal line segments. The output consists of several planar patches each lying in one of the two planes and exactly $10,000$ lines, such that each contains one intersection point in one plane and one in the other plane.

Lines	Planar Curves	Spherical Arcs	Planar Regions	Time (secs)
10,000	36	1,224	17,060	20.74

The table to the right lists all output elements. 703 arrangements in the plane and 200 arrangements on the sphere were constructed during the process. All single output lines are represented by vertices of arrangements on the sphere. Such an arrangement is depicted in the figure above. The origin of the sphere is the intersection point of two line segments S_1 and S_{40} lying in the same plane. The arrangement is induced by the point set $\{\Xi_{S_1 \cap S_{40}}(S_i) \,|\, i = 2, \ldots, 39\}$.



The figure to the right depicts an arrangement in the plane constructed during the process. The arrangement is induced by the point set $\{\Psi_{S_1 S_{39}}(S_i) \,|\, i = 2, \ldots, 38\}$. The line segments S_1 and S_{39} are parallel segments lying in the same plane. Each face of the arrangement represents a ruled surface patch, such that each line lying in the surface intersects at least 6 and up to 20 line segments. Different colors represent different number of generating line segments.

5.2 Transformed Grid

We have conducted three more experiments using a transformed version of the Grid data set. First, we slightly perturbed the input line segments, such that every two line segments became skew and the directions of every three line segments became linearly independent (referred to as **PG**). Secondly, we translated

Input	Unlimited Precision			Double Precision	
	Time (secs)		Lines	Time	Lines
	LTS	Redburn		Redburn	
PG	23.72	140.17	12,139	0.70	12,009
TG 1	11.83	132.80	5,923	0.69	5,927
TG 2	6.90	128.80	1,350	0.70	1,253

the (perturbed) horizontal line segments of one grid along the plane that contains this grid (referred to as **TG 1**), increasing the distance between the (perturbed) vertical and horizontal line segments of that grid. This drastically reduced the number of output lines. Thirdly, we translated the (perturbed) horizontal line segments of the other grid along the plane that contains this grid (referred to as **TG 2**), further reducing the number of output lines. The table above shows the number of output lines and the time it took to perform the computation using our implementation, referred to as **LTS**. The monotonic relation between the output size and time consumption of our implementation is prominent. The table also shows the time it took to perform the computation using two instances of a program developed by J. Redburn [15], which represents lines by their Plücker coordinates and exhaustively examines every quadruple of input line segments. One instance, relies on a number type with unlimited precision, while the other resorts to double-precision floating-point numbers. As expected, when limited precision numbers were used, the output was only an approximation. Notice that the influence of the output size on the time consumption of Redburn's implementation is negligible.[6]

5.3 Random Input

Input	Time (secs)			Lines
	LTS	LLTS	Redburn	
Short	3.04	1.06	300.4	0
Medium	6.80	2.82	314.0	20,742
Long	12.36	5.15	327.0	64,151

The Random data set consists of 50 line segments drawn uniformly at random. In particular, the endpoints are selected uniformly at random within a sphere. We experimented with three different radii, namely, **Short**, **Medium**, and **Long** listed in increasing lengths. We verified that the line segments are in general position; that is, the directions of every three are linearly independent and they are pairwise skew. The table above shows the number of output lines and the time it took to perform the computation using (i) our implementation referred to as **LTS**, (ii) our implementation enhanced with the lazy mechanism referred to as **LLTS** (see Section 4), and (iii) the instance of Redburn's implementation that relies on unlimited precision. Once again, one can clearly observe how the time consumption of our implementation decreases with the decrease of the output size, which in turn decreases with the decrease in the line-segment lengths. Adversely, the time consumption of Redburn's implementation hardly changes.

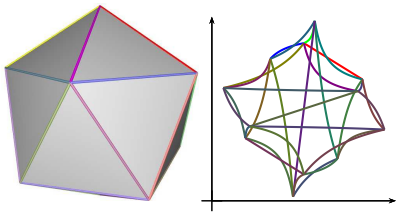
6 Future Work

We are, in particular, motivated by assembly-partitioning problems, where a given collection of pairwise interior disjoint polyhedra in some relative position

[6] Our attempts to run Redburn's code on degenerate input failed, as it seems not to well handle such cases; thus, we were unable to experiment with the original Grid data set using this implementation.

in $\mathbb{R}^3$, referred to as an assembly, has to be partitioned into its basic polyhedra through the applications of a sequence of transforms applied to subsets of the assembly [11, 21]. The LTP problem is a building block in a solution that we forsee to certain assembly-partitioning problems. We strive to develop an output-sensitive algorithm that solves the LTP problem, and provide an exact implementation of it. We have already conceived the general framework for such an algorithm and implemented a raw version that handles the general position case. However, we still need to enhance the algorithm and its implementation to handle all cases and carefully analyze them.

A glimpse at this future development can be seen in the figure to the right. It shows an icosahedron P and the arrangement induced by the point set $\Psi_{S_1 S_2}(E(P))$, where $E(P)$ is the set of the edges of P, and S_1 and S_2 are two skew segments (omitted in the figure). The color of each edge of the arrangement is the same as the color of its generating icosahedron edge. The boundary of the hole in the unbounded face contains points that represent lines that intersect S_1 and S_2 and are tangent to P.



7 Acknowledgement

We thank Michael Hoffmann for helpful discussions on assembly partitioning, which inspired us to conduct the research discussed in this article. We also thank Linqiao Zhang who provided us with Redburn's code that was used for the experiments. Zhang used it as part of an implementation of an algorithm that constructs the visibility skeleton [23].

References

1. E. Berberich, E. Fogel, D. Halperin, M. Kerber, and O. Setter. Arrangements on parametric surfaces II: Concretizations and applications. *Math. in Comput. Sci.*, 4:67–91, 2010.
2. E. Berberich, E. Fogel, D. Halperin, K. Mehlhorn, and R. Wein. Arrangements on parametric surfaces I: General framework and infrastructure. *Math. in Comput. Sci.*, 4:45–66, 2010.
3. E. Berberich, M. Hemmer, and M. Kerber. A generic algebraic kernel for non-linear geometric applications. In *Proc. 27th Annu. ACM Symp. Comput. Geom.*, pages 179–186, New York, NY, USA, 2011. ACM Press.
4. H. Brönnimann, C. Burnikel, and S. Pion. Interval arithmetic yields efficient dynamic filters for computational geometry. *Disc. Appl. Math.*, 109:25–47, 2001.
5. H. Brönnimann, O. Devillers, V. Dujmovic, H. Everett, M. Glisse, X. Goaoc, S. Lazard, and H. suk Na. Lines and free line segments tangent to arbitrary three-dimensional convex polyhedra. *SIAM J. on Computing*, 37:522–551, 2006.
6. H. Brönnimann, H. Everett, S. Lazard, F. Sottile, and S. Whitesides. Transversals to line segments in three-dimensional space. *Disc. Comput. Geom.*, 34:381–390, 2005. 10.1007/s00454-005-1183-1.

7. J. Demouth, O. Devillers, H. Everett, M. Glisse, S. Lazard, and R. Seidel. On the complexity of umbra and penumbra. *Comput. Geom. Theory Appl.*, 42:758–771, 2009.
8. O. Devillers, M. Glisse, and S. Lazard. Predicates for line transversals to lines and line segments in three-dimensional space. In *Proc. 24th Annu. ACM Symp. Comput. Geom.*, pages 174–181. ACM Press, 2008.
9. O. Devillers and S. Pion. Efficient exact geometric predicates for Delaunay triangulations. In *Proc. 5th Workshop Alg. Eng. Experiments*, pages 37–44, 2003.
10. H. Everett, S. Lazard, W. Lenhart, J. Redburn, and L. Zhang. On the degree of standard geometric predicates for line transversals. *Comput. Geom. Theory Appl.*, 42(5):484–494, 2009.
11. E. Fogel and D. Halperin. Polyhedral assembly partitioning with infinite translations or the importance of being exact. In H. Choset, M. Morales, and T. D. Murphey, editors, *Alg. Foundations of Robotics VIII*, volume 57 of *Springer Tracts in Advanced Robotics*, pages 417–432. Springer, Heidelberg, Germany, 2009.
12. M. Hemmer, S. Hert, L. Kettner, S. Pion, and S. Schirra. Number types. In *CGAL User and Reference Manual*. CGAL Editorial Board, 4.0 edition, 2012. `http://www.cgal.org/Manual/4.0/doc_html/cgal_manual/packages.html#Pkg:NumberTypes`.
13. M. McKenna and J. O'Rourke. Arrangements of lines in 3-space: a data structure with applications. In *Proc. 4th Annu. ACM Symp. Comput. Geom.*, pages 371–380, New York, NY, USA, 1988. ACM Press.
14. S. Pion and A. Fabri. A Generic Lazy Evaluation Scheme for Exact Geometric Computations. *Sci. Comput. Programming*, 76(4):307–323, Apr 2011.
15. J. Redburn. *Robust computation of the non-obstructed line segments tangent to four amongst n triangles.* PhD thesis, Williams College, Massachusetts, 2003.
16. O. Salzman, M. Hemmer, B. Raveh, and D. Halperin. Motion planning via manifold samples. In *Proc. 19th Annu. Eur. Symp. Alg.*, pages 493–505, 2011.
17. S. Teller and M. Hohmeyer. Determining the lines through four lines. *j. of graphics, gpu, and game tools*, 4(3):11–22, 1999.
18. The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 4.0 edition, 2012. `http://www.cgal.org/Manual/4.0/doc_html/cgal_manual/title.html`.
19. R. Wein, E. Fogel, B. Zukerman, and D. Halperin. Advanced programming techniques applied to CGAL's arrangement package. *Comput. Geom. Theory Appl.*, 38(1–2):37–63, 2007. Special issue on CGAL.
20. R. Wein, E. Fogel, B. Zukerman, and D. Halperin. 2D arrangements. In *CGAL User and Reference Manual*. CGAL Editorial Board, 4.0 edition, 2012. `http://www.cgal.org/Manual/4.0/doc_html/cgal_manual/packages.html#Pkg:Arrangement2`.
21. R. H. Wilson, L. Kavraki, J.-C. Latombe, and T. Lozano-Pérez. Two-handed assembly sequencing. *Int. J. of Robotics Research*, 14:335–350, 1995.
22. C.-K. Yap and T. Dubé. The exact computation paradigm. In D.-Z. Du and F. K. Hwang, editors, *Computing in Euclidean Geometry*, volume 1 of *LNCS*, pages 452–492. World Scientific, Singapore, 2nd edition, 1995.
23. L. Zhang, H. Everett, S. Lazard, C. Weibel, and S. Whitesides. On the size of the 3D visibility skeleton: Experimental results. In *Proc. 16th Annu. Eur. Symp. Alg.*, volume 5193/2008 of *LNCS*, pages 805–816, Karlsruhe Allemagne, 2008. Springer.