Lines Through Segments in Three Dimensional Space*

(Extended Abstract)

Efi Fogel† Michael Hemmer† Asaf Porat† Dan Halperin†

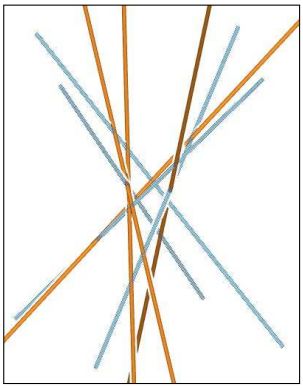
Abstract

Given a set $\mathcal{S}$ of n line segments in three-dimensional space, finding all the lines that simultaneously intersect at least of line segments in $\mathcal{S}$ is a fundamental problem that arises in a variety of domains including computer graphics, computer vision, robotics and automation, to mention a few. We refer to this problem as *the lines-through-segments problem*, or LTS for short. We present an efficient output-sensitive algorithm and its exact implementation to solve the LTS problem. The algorithm properly handles all degenerate cases. For example, a line segment may degenerate to a point, several segments may be coplanar, parallel, concurrent, collinear, or they can even overlap. We provide a detailed analysis of all the (degenerate) cases that can arise. To the best of our knowledge, this is the first algorithm (and 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`.

*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.

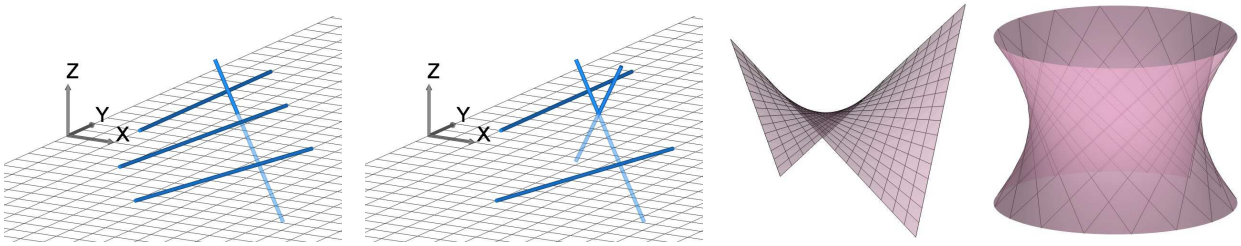
†School of Computer Science, Tel-Aviv University, 69978, Israel. `efifogel@gmail.com`, `mhsaar@googlemail.com`, `asafpor@gmail.com`, `danha@post.tau.ac.il`.

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 four line segments in $\mathcal{S}$. The figure to the right depicts four lines (drawn in green) that intersect four line segments (drawn in blue with a halftone pattern).

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 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, vision, and robotics, such as computing umbra and penumbra cast by a light source [9]. We are, in particular, motivated by assembly-partitioning problems, where a given collection of pairwise interior disjoint polyhedra in some relative position 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 [13, 20].



(a) Three coplanar line segments lying in a plane P, and an additional line segment piercing P.

(b) Two coplanar line segments lying in a plane P, and two additional line segments piercing P at the same point.

(c) A hyperbolic paraboloid.

(d) A hyperboloid of one sheet.

Figure 1: Configurations of lines segments in which an infinite number lines intersect four line segments.

The number of lines that intersect four lines in $\mathbb{R}^3$ is 0, 1, 2, or infinite. Brönnimann et al. [7] showed that the number of lines that intersect four arbitrary line segments in $\mathbb{R}^3$ is 0, 1, 2, 3, 4, or infinite. The latter may happen only if the segments lie in one of the following configurations:[1] (i) The four line segments are coplanar. (ii) Three lines segments lie in the same plane P, which is pierced by the fourth segment; see Figure 1a. (iii) Two line segments lie in the same plane P, while the other two pierce P at the same point; see Figure 1b. (iv) At least three line segments intersect at the same point. (v) At least two line segments overlap. (vi) All four line segments are contained in the same ruling of a *hyperbolic paraboloid* or a *hyperboloid of one sheet*; see Figure 1c and Figure 1d, respectively. In addition, Brönnimann et al. showed that the lines lie in at most four maximal *connected components*.[2]

A straightforward method to find all the lines that intersect four lines, given a set of n lines, examines each quadruplet of lines. The examination is simplified using the Plücker coordinate

[1] Some conditions are omitted, e.g., no pair of the line segments are collinear.

[2] 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.

representation. The Plücker coordinates of a line L, defined by a sample point p on the line and a vector $\vec{u}$ that expresses the direction of the line, are the six-tuple $< \vec{u}, \vec{u} \times p >$. The *side product* of two lines L_a and L_b with Plücker coordinates $a = [a_1, \ldots, a_6]$ and $a = [b_1, \ldots, b_6]$, is defined as [18]: $a \odot b = (a_1b_4 + a_2b_5 + a_3b_6 + a_4b_1 + a_5b_2 + a_6b_3)$. The side product is zero whenever L_a and L_b intersect or are parallel and non zero otherwise. The method of finding intersecting lines using the Plücker coordinates representation has been used by Hohmeyer and Teller [18] and also described by Redburn [16]. This method was later used by Everett et al. [12] as a building block for the problem of finding line transversals (the set of lines that intersect all given line segments). The use of Plücker coordinates simplifies the algebra but does not obviate the need to process each quadruplet of lines. The running time of this method is $O(n^4)$.

The combinatorial complexity of all 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 lengths of the input line segments.

We present an efficient output-sensitive algorithm, and its complete and robust implementation that solves the LTS problem in three-dimensional Euclidean space. 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 algorithm (and 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 [14] to represent the set of lines that intersect three lines as a rectangular hyperbola with vertical and 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, 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 $\mathbb{R}^3$ [7].

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. [6] 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. [10] 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) [**?**]. The implementation is mainly based on the *2D Arrangements* package of the library [**?**]. 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 [3,19], 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 [2]. We plan to make our new component available as part of a future public release of CGAL.

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 deferred to Appendix A and Appendix B, respectively.

2 Representation

For two fixed line segments S_1 and S_2 this section discusses the encoding of all lines that intersect S_1 and S_2 and intersect 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 $a = 0$ 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. Given two lines L_1 and L_2 we define a map $\Psi_{L_1L_2}$ as follows:

$$\Psi_{L_1L_2}(q) = \{(t_1, t_2) \in \mathbb{R}^2 \,|\, L_1(t_1),\ L_2(t_2),\ \text{and } q \text{ are collinear}\}\,.$$

That is, $\Psi_{L_1L_2}(q)$ maps a point in $\mathbb{R}^3$ to a set in $\mathbb{R}^2$. This set, 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 a line, namely, 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 $\Psi_{s_1s_2}(q) = \{[0,1]^2, |\, S_1(t_1),\ S_2(t_2),\ \text{and } q \text{ are collinear}\}$ 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. A complete analysis of $\Psi_{L_1L_2}(S_3)$, though, is necessary in order to handle all cases. However, due to limited space, we concentrate on the case where the L_1, L_2, and S_3 are pairwise skew, and the directions of L_1, L_2 and $L(S_3)$ are linearly independent, and defer the complete characterization to Appendix A. Section 2.2 introduces an additional mapping necessary in case S_1 and S_2 intersect.

2.1 Directions Are Linearly Independent

In this section we discuss all cases in which the direction vectors of the underlying lines of the segments are linearly independent. In this setting we can always apply a rational affine transformation such that the three segments are given by $S_i(t_i) = p_i + t_i \cdot d_i,\ i \in \{1, 2, 3\}$, where $p_1 = (a, b, c)$, $p_2 = (d, e, f)$, $p_3 = \mathcal{O}$ and $d_i = e_i$ (where e_i denotes the unit vector along the ith axis). Thus, we continue with a refined case distinction that only depends on the coordinates of p_1 and p_2.

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

$\mathbf{b \neq 0}$, $\mathbf{d \neq 0}$, and $\mathbf{c \neq f}$: All three lines are pairwise skew. Consider the points $L_1(t_1)$, $L_2(t_2)$, and $L_3(t_3)$. These points are collinear iff

$$|(L_1(t_1) - L_2(t_2)) \times (L_3(t_3) - L_2(t_2))| = 0\,. \qquad (2.1)$$

These are three dependent equations in three unknowns. Eliminating t_3 we obtain the following expression for t_2 in terms of t_1:

$$t_2(t_1) = \frac{e \cdot t_1 + (a \cdot e - d \cdot b)}{t_1 + a}\,. \qquad (2.2)$$

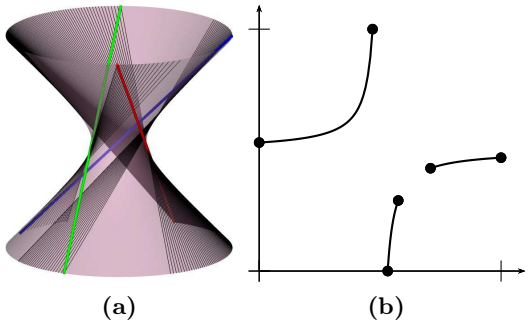


Figure 2: (a) Three surface patches the lines of which intersect three skew line segments, S_1, S_2, and S_3, in $\mathbb{R}^3$. These surface patches are contained in a hyperboloid of one sheet. (b) The point set $\Psi_{S_1S_2}(S_3)$.

It implies that $\Psi_{L_1L_2}(L_3)$ is a rectangular hyperbola with a vertical asymptote at $t_1 = -a$ and a horizontal asymptote at $t_2 = -e$. The point $(d - a, b - e)$ corresponds to the line that is parallel to L_3 and (by definition) intersects L_1 and L_2. Thus, this point is not in $\Psi_{L_1L_2}(L_3)$, as we consider affine space. Nonetheless, we are interested in $\Psi_{L_1L_2}(S_3)$, where $S_3 = \{L_3(t_3) \,|\, t_3 \in [0,1]\}$. Solving the system of equation 2.1 for t_1 in terms of t_3 yields

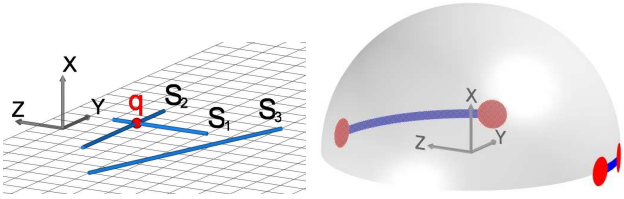
$$t_1(t_3) = \frac{(d-a)t_3 + fa - dc}{t_3 - f}\,. \qquad (2.3)$$

As t_3 is restricted to $[0,1]$, t_1 is restricted to $\mathcal{T} = \{t_1(t_3) \,|\, t_3 \in [0,1]\}$. $\Psi_{L_1L_2}(S_3)$ is not defined for values of $t_1 \notin \mathcal{T}$. Let $t' = \min(t_1(0), t_1(1))$ and $t'' = \max(t_1(0), t_1(1))$, where $t_1(0) = (dc - af)/f$ and $t_1(1) = (dc + a - fa - d)/(f - 1)$. $t_1(t_3)$ is a hyperbola with a vertical asymptote at $t_3 = f$. If $f \in [0,1]$, then $\mathcal{T} = (-\infty, t'] \cup [t'', \infty)$. Otherwise, $\mathcal{T} = [t', t'']$. Recall that $\Psi_{L_1L_2}(S_3)$ is also not defined for the value $t_1 = -a$ due to the vertical asymptote of $t_2(t_1)$. It follows that $\Psi_{S_1S_2}(S_3)$ consists of at most three maximal connected components, where each component represents a patch of a ruled surface as depicted in Figure 2.

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 $\mathcal{O}\}$.

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 to the right 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, as explained in Appendix B.

Corollary 2.1. *$\Psi_{S_1S_2}(S_3) \subset \mathbb{R}^2$ is either a point, a one dimensional set consisting of line segments or arcs of rectangular hyperbolas with horizontal and vertical asymptotes, or a two-dimensional set bounded by linear segments or arcs of such hyperbolas.*

We are now ready to describe our algorithm for solving the LTS problem in its full generality. For further details related to the variant cases handled see Appendix A.

3 The Algorithm

The input is a set $\mathcal{S} = \{S_1, \ldots, S_n\}$ of n line segments in $\mathbb{R}^3$. In general an input line segment imposes an intersection constraint. By default we assume that a sub-segment that is the intersection of multiple overlapping line segments imposes a single constraint, and a point that is either the intersection of multiple line segments, or simply a degenerate line segment, imposes two constraints. The user can override the default setting and require that every input line segment imposes exactly a single constraint. The output is a set of at most $O(n^4)$ (one-dimensional) lines or (two-dimensional) ruled surface patches in $\mathbb{R}^3$, such that each line abides by exactly four intersection constraints imposed by the line segments in $\mathcal{S}$, and all lines of each ruled surface patch abide by exactly four such intersection constraints. The line segments that impose the constraints of an output element are referred to as the *generating line segments* of that element. The generating line segments of every output surface patch or line are provided as part of the output. An element of the output is thus a pair of a line or a surface patch together with a quadruple of generating line segments.

To simplify the exposition of the algorithm, we assume that the line segments are full-dimensional, pairwise disjoint, and no three line segments are coplanar. We describe the algorithm that handles this case. Then, we relax the assumption to allow the line segments to intersect pairwise at discrete and distinct points though. We describe the adjustments to the original algorithm necessary to handle the additional case. Due to limited space we defer the description of the complete algorithm that handles all cases to Appendix B. The complete algorithm also respects several different settings selected by the user. They are also listed in the appendix.

3.1 Input Line Segments Are Pairwise Disjoint

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 line 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 in $\mathcal{S} = \{S_1, \ldots, S_n\}$.

```
1  for i = 1, ..., n − 3,
2      for j = n, ..., i + 3,
3          Construct the 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}.
```

In Line 3 of Algorithm 1 we construct the arrangement $\mathcal{A}_{S_iS_j}$ induced by the point set $\mathcal{C}_{ij} = \{\Psi_{S_iS_j}(S_k) \,|\, k = i+1, \ldots, j-1\}$. We process the line segments $S_{i+1}, \ldots, S_{j-1}$ one at a time to produce the inducing point 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.

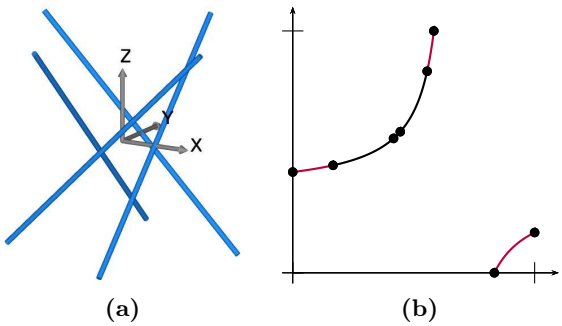


Figure 3: (a) Four line segments, S_1, S_2, S_3, S_4, supported by four lines of one ruling of a *hyperbolic paraboloid*, respectively; see also Figure 1c. (b) The arrangement $A_{S_1S_2}$. The edge drawn in purple is induced by two overlapping curves, one in $\Psi_{S_1S_2}(S_3)$ and the other in $\Psi_{S_1S_2}(S_4)$.

We store only the minimal necessary set of line segments in every sequence to save space and clearly distinguish between line segments that map to (zero-dimensional) points and those that map to (one-dimensional) curves. A member of a sequence of a vertex is a line segment that maps to a point $p \in \mathcal{C}_{ij}$. A member of a sequence of an edge is a line segment that maps to a curve $C \in \mathcal{C}_{ij}$. Consider a curve $C \in \mathcal{C}_{ij}$, such that $C \in \Psi_{S_iS_j}(S)$. The sequences of all edges induced by C contain S. However, the sequences of line segments of all vertices incident to these edges do not contain S, as this information is immediately accessible from the incident edges.

Curves in $\mathcal{C}_{ij}$ may overlap; see Figure 3b. This degeneracy is handled as follows: let $\mathcal{A}'_{S_1S_2}$ denote an intermediate arrangement during the plane sweep. Let C' denote the geometric embedding of an existing edge e' of the arrangement $\mathcal{A}'_{S_1S_2}$, and let $C'' \in \Psi_{S_iS_j}(S)$ denote a new curve being inserted into the arrangement, such that C'' and C' overlap; let C denote the common subcurve, and let e denote the newly created edge whose geometric embedding is C. We store with the edge e a copy of the sequence of line segments stored in e' and insert S into it.

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 segments; see Appendix A. The 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.

In Line 4 of Algorithm 1 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. The eligibility of a given cell is immediately established from the sequence of line segments stored in that 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. By default we consider a surface patch of the output represented by an edge or a face open. For example, consider an edge e that satisfies the output criteria, and let C denote its geometric embedding. The curve C is provided to the user as part of the iteration over the one-dimensional output elements. The two endpoints of C are provided to the user as part of the iteration over the zero-dimensional output elements. The user can override this setting, and choose to consider surface patches closed. In this case the iteration over the zero-dimensional output elements results with only eligible vertices that are not incident to eligible edges.

3.2 Input Line Segments May Intersect at Discrete and Distinct Points

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 arrangements constructed by Algorithm 1. 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 $\{\Psi_{S_iS_j}(S) \,|\, S \in \mathcal{S} \setminus \{S_i, S_j\}$. Let S_k and S_ℓ be two additional input line segments, such that there exists a line, L, that intersects both S_k and S_ℓ and contains p. Our assumption that the line segments are not concurrent assures that neither S_k nor S_ℓ contains p. L is represented by cells in five different arrangements—all arrangements indexed by unordered pairs taken from $\{i, j, k, \ell\}$ excluding (i, j). In order to output L only once, we must filter out all cells but one. While this modification does not increase the asymptotic complexity of the algorithm, our experiments show a considerable degradation in performance. Instead, we resort to a more efficient solution that also uses two-dimensional arrangements, but this time on the sphere; see Algorithm 2 for the pseudo code. Nevertheless, we support the user option to exhaustively construct arrangements for all unordered pairs, in which case we obtain maximally connected components; see Appendix B.5.

Algorithm 2 Compute lines that intersect line segments in $\mathcal{S} = \{S_1, \ldots, S_n\}$.

```
1  for i = 1, ..., n − 3,
2      for j = n, ..., i + 3,
3          Construct the arrangement A_{S_iS_j} induced by {Ψ_{S_iS_j}(S_k) | k = i + 1, ..., j − 1}.
4          Extract lines that intersect S_i and S_j from A_{S_iS_j}.
5          if S_i and S_j intersect,
6              Construct the 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 6 of Algorithm 2 we construct an arrangement on the sphere centered at p—the intersection point of S_i and S_j. The arrangement is induced by the point set $\mathcal{C}^s_{ij} = \{\Xi_{S_i\cap S_j}(S_k) \,|\, k = i+1, \ldots, j-1\}$. 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 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.

As with the planar arrangements, we store only the minimal necessary set of line segments in every sequence. A member of a sequence of a vertex is a line segment that maps to a (zero-dimensional) point $p \in \mathcal{C}^s_{S_i\cap S_j}$. A member of a sequence of an edge is a line segment that maps to a (one-dimensional) geodesic arc $C \in \mathcal{C}^s_{S_i\cap S_j}$.

We extract the information from the arrangements on the sphere and provide it to the user in a usable format. All the settings that apply to the processing of the arrangements in the plane apply to the processing of the arrangements on the sphere as well; see Section 3.1. As with the aforementioned processing of the arrangements in the plane, we provide the user with the ability to iterate over eligible vertices and over eligible edges separately.

4 Lazy Algebraic Tools

Our implementations are exact and complete. In order to achieve this, CGAL in general, and the CGAL *2D Arrangements* package in particular, follows the *exact geometric-computation (EGC) paradigm*. 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, as it is possible to use fast inexact arithmetic (e.g., double-precision floating-point arithmetic [11]), 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 [15] 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 [5], 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* [1], 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 class [**?**] is written such that it takes a *traits* class as a template argument, which defines the used type of curves and also provides the required operations on these curves.

For the arrangement of geodesic arcs on the sphere we use the existing and efficient traits class that we have used before [2]. As this only requires a linear kernel, it uses CGAL's efficient Lazy Kernel [5]. 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 [17]. The class uses the general univariate algebraic kernel [4] of CGAL, which does not offer lazy constructions.

The aforementioned traits class is capable of representing rectangular hyperbolic arcs with horizontal and vertical asymptotes. However, since it was developed for general rational functions, the code is written assuming arbitrary degree in the numerator and denominator polynomials of the rational function. In our case the degree of both is just one. It follows that the degree of the polynomial, the roots of which represent the x-coordinates of intersection points of two hyperbolas, is at most 2. That is, the solution to these polynomials, and thus the coordinates of the intersection points, are numbers of only algebraic degree 2. The *square-root extension* type of CGAL represents such a number as $a + b\sqrt{c}$, where $a, b, c \in \mathbb{Q}$. This explicit representation makes it possible to use the number type in conjunction with the lazy mechanism.

In order to benefit from this framework, we implemented a univariate algebraic kernel that is, similar to [8], restricted to polynomials of degree 2. Enabling the use of CGAL's *square-root extensions* and in particular to use the lazy mechanism to speed up the computation. In addition, we enhanced the implementation of the existing rational function traits, such that it is capable of using any algebraic kernel that complies with the requirements in [**?**]. This traits class, instantiated with the new algebraic kernel, uses lazy constructions and is thus able to handle hyperbolic arcs with horizontal and vertical asymptotes in a more efficient manner than the one presented in [17], as shown by experiments in Section 5.

5 Experimental Results

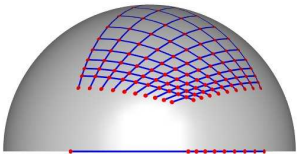
We have conducted several experiments on three types of data sets. The first produces the worst case combinatorial output and has many degeneracies. The second consists of transformed versions of the first and has many near-degeneracies. The third comprises random input. We report on the time consumption 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

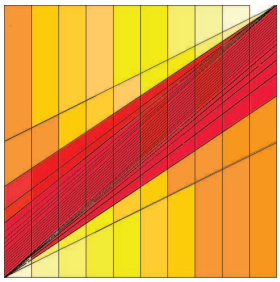
The Grid data set comprises 40 line segments arranged in two grids of 20 lines segments each lying in two planes parallel to the yz-plane; see Section 1. Each grid consists of ten vertical and ten horizontal line segments. The output consists of several planar patches each lying in one of the two planes and exactly $10,000$ lines, such that each contains one intersection point in one plane and one in the other plane. Table 1 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 at the bottom of the previous page. 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 $\{\Xi_{S_1 \cap S_{40}}(S_i) \,|\, i = 2, \ldots, 39\}$.

Table 1: Output for the Grid Input. Time is measured in seconds.

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



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 originating line segments.



5.2 Transformed Grid

We conducted three additional 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 **Perturbed Grid**). Secondly, we translated the (perturbed) horizontal line segments of one grid along the plane that contains this grid (referred to as **Perturbed Grid 1**), increasing the distance between the (perturbed) vertical and horizontal line segments of that grid. This drastically reduced the number of

Table 2: Perturbed Grid. Time is measured in seconds.

Input	Unlimited Precision			Double Precision	
	Time		Lines	Time	Lines
	LTS	Redburn		Redburn	
Perturbed Grid	23.72	140.17	12,139	0.70	12,009
Translated Grid 1	11.83	132.80	5,923	0.69	5,927
Translated Grid 2	6.90	128.80	1,350	0.70	1,253

output lines. Thirdly, we translated the (perturbed) horizontal line segments of the other grid along the plane that contains this grid (referred to as **Perturbed Grid 2**), further reducing the number of output lines. Table 2 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 [16], 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.[4]

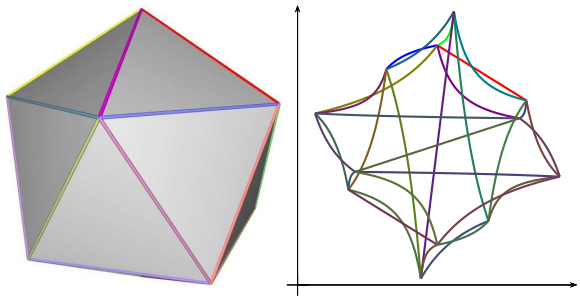
5.3 Random Input

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. Table 3 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.

Table 3: Random Input, 50 segments.

Input	Time			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

6 Future Work



As mentioned in the introduction section, we are motivated by assembly-partitioning problems. The LTP problem is a building block in a solution that we forsee to certain assembly-partitioning problems. We believe that the case analysis of lines that intersect three line segments presented in Section 2 could also serve as the theoretical foundation of an output-sensitive algorithm that solves the LTP problem. 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 develop-

[4]Redburn's implementation does not handle well degenerate input. Thus, we were unable to experiment with the original Grid data set using this implementation.

ment can be seen in the figure above. It shows an icosahedron P and the arrangement induced by the point set $\Psi_{S_1S_2}(E(P))$, 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 [21].

References

[1] M. H. Austern. *Generic Programming and the STL*. Addison-Wesley, Boston, MA, USA, 1999.

[2] 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.

[3] 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.

[4] 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.

[5] H. Brönnimann, C. Burnikel, and S. Pion. Interval arithmetic yields efficient dynamic filters for computational geometry. *Disc. Appl. Math.*, 109:25–47, 2001.

[6] 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.

[7] 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.

[8] P. M. de Castro, F. Cazals, S. Loriot, and M. Teillaud. Design of the CGAL 3D spherical kernel and application to arrangements of circles on a sphere. *Comput. Geom. Theory Appl.*, 42(6–7):536–550, 2009.

[9] 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.

[10] 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.

[11] O. Devillers and S. Pion. Efficient exact geometric predicates for Delaunay triangulations. In *Proc. 5th Workshop Alg. Eng. Experiments*, pages 37–44, 2003.

[12] 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.

[13] 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.

[14] 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.

[15] S. Pion and A. Fabri. A Generic Lazy Evaluation Scheme for Exact Geometric Computations. *Sci. Comput. Programming*, 76(4):307–323, Apr 2011.

[16] J. Redburn. *Robust computation of the non-obstructed line segments tangent to four amongst n triangles*. PhD thesis, Williams College, Massachusetts, 2003.

[17] 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.

[18] S. Teller and M. Hohmeyer. Determining the lines through four lines. *j. of graphics, gpu, and game tools*, 4(3):11–22, 1999.

[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. H. Wilson, L. Kavraki, J.-C. Latombe, and T. Lozano-Pérez. Two-handed assembly sequencing. *Int. J. of Robotics Research*, 14:335–350, 1995.

[21] 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.

A Characterization of Degenerate Cases

This appendix provides the complete characterization of $\Psi_{L_1L_2}$; see Section 2.

A.1 Directions Are Linearly Independent

$\mathbf{b \neq 0},\ \mathbf{d \neq 0}$, and $\mathbf{c \neq f}$: All three lines are pairwise skew. $\Psi_{L_1L_2}(L_3)$ is a rectangular hyperbola with a vertical asymptote at $t_1 = -a$ and a horizontal asymptote at $t_2 = -e$; see Figure 2. The point $(d-a, b-e) \notin \Psi_{L_1L_2}(L_3)$ corresponds to the line that intersects L_1 and L_2 but is parallel to L_3. Nonetheless, we are interested in $\Psi_{L_1L_2}(S_3)$, where $S_3 = L_3(t_3)$ defined over $t_3 \in [0,1]$. As t_3 is restricted to $[0,1]$, t_1 is restricted to $\mathcal{T} = \{t_1(t_3) \,|\, t_3 \in [0,1]\}$. $\Psi_{L_1L_2}(S_3)$ is not defined for values of $t_1 \notin \mathcal{T}$. Let $t' = \min(t_1(0), t_1(1))$ and $t'' = \max(t_1(0), t_1(1))$, where $t_1(0) = (dc - af)/f$ and $t_1(1) = (dc + a - fa - d)/(f-1)$. $t_1(t_3)$ is a hyperbola with a vertical asymptote at $t_3 = f$. If $f \in [0,1]$, then $\mathcal{T} = (-\infty, t'] \cup [t'', \infty)$. Otherwise, $\mathcal{T} = [t', t'']$. Recall that $\Psi_{L_1L_2}(S_3)$ is also not defined for the value $t_1 = -a$ due to the vertical asymptote of $t_2(t_1)$; see Figure 2.
$\mathbf{b \neq 0},\ \mathbf{d \neq 0}$, and $\mathbf{c = f}$: L_1 and L_2 intersect at $p = (d, b, c) = (d, b, f)$. L_3 is skew to both and intersects the $z = c$ plane (which is spanned by L_1 and L_2) at $q = (0, 0, c) = (0, 0, f)$. As in the previous case, $\Psi_{L_1L_2}(L_3)$ is a rectangular hyperbola. However, the point $(d-a, b-e) \in \Psi_{L_1L_2}(L_3)$ represents all lines containing p and intersecting L_3; see also Section 2.2. In case $q \notin S_3$, $\Psi_{L_1L_2}(S_3)$ degenerates to $\{(d-a, b-e)\}$.
$\mathbf{b = 0},\ \mathbf{d \neq 0}$, and $\mathbf{c \neq f}$: L_1 intersects L_3 at $p = (0, 0, c)$. L_2 intersects the xz-plane at $q = (d, 0, f)$. Eliminating t_3 from 2.1 we obtain $(e + t2)(a + t1) = 0$. Thus, $\Psi_{L_1L_2}(L_3)$ is a vertical line at $t1 = -a$ and a horizontal line at $t_2 = -e$. $t1 = -a$ represents all lines containing p and L_2. $t_2 = -e$ represents all lines in the xz-plane that contain q. Since we consider affine space, the point $(d-a, -e) \notin \Psi_{L_1L_2}(L_3)$. The point set $\Psi_{L_1L_2}(S_3)$ (i) includes the vertical line only if $p \in S_3$, and (ii) includes the horizontal line only if $q \in S_2$. As t_3 is restricted to $[0,1]$, t_1 is restricted to $\mathcal{T} = \{t_1(t_3) \,|\, t_3 \in [0,1]\}$. The horizontal line is not defined for values of $t_1 \notin \mathcal{T}$. Let $t' = \min(t_1(0), t_1(1))$ and $t'' = \max(t_1(0), t_1(1))$, where $t_1(0) = (dc - af)/f$ and $t_1(1) = (dc + a - fa - d)/(f-1)$. $t_1(t_3)$ is a hyperbola with a vertical asymptote at $t_3 = f$. If $f \in [0,1]$, then $\mathcal{T} = (-\infty, t'] \cup [t'', \infty)$. Otherwise, $\mathcal{T} = [t', t'']$. For symmetry reasons this case essentially also covers the case $\mathbf{b \neq 0},\ \mathbf{d = 0}$, where the characters of the vertical and horizontal lines exchange.
$\mathbf{b = 0},\ \mathbf{d = 0}$, and $\mathbf{c \neq f}$: L_1 and L_2 are pairwise skew and intersect L_3 at $p = (0, 0, c)$ and $q = (0, 0, f)$, respectively. $\Psi_{L_1L_2}(L_3)$ consists of a vertical line at $t_1 = -a$ and a horizontal line at $t_2 = -e$. $\Psi_{L_1L_2}(S_3)$ includes the vertical and horizontal lines if S_3 contains $p = (0, 0, c)$ and $q = (0, 0, f)$, respectively.
$\mathbf{b = 0},\ \mathbf{d \neq 0}$, and $\mathbf{c = f}$: L_1 and L_2 intersect at $p = (d, 0, c) = (d, 0, f)$. L_1 and L_3 intersect at $q = (0, 0, c) = (0, 0, f)$. $\Psi_{L_1L_2}(L_3)$ consists of a vertical line at $t_1 = -a$ and a horizontal line at $t_2 = -e$. The latter is included in $\Psi_{L_1L_2}(S_3)$ if $q \in S_3$. $t_1 = -a$ corresponds to all lines containing q and intersect L_2. All points on $t_2 = -e$ correspond to L_1. For symmetry reasons this essentially also covers the case $\mathbf{b \neq 0},\ \mathbf{d = 0}$.
$\mathbf{b = 0},\ \mathbf{d = 0}$, and $\mathbf{c = f}$: The three lines are concurrent at $p = (0, 0, c) = (0, 0, f)$. $\Psi_{L_1L_2}(L_3) = \{(-a, -e)\}$. $(-a, -e)$ represents all lines that contain p.

A.2 Directions Are Not Linearly Independent

A.2.1 Directions of L_1 and L_2 are Linearly Independent

We consider the case where L_1 and L_2 are linearly independent. Thus, we can assume that $d_1 = e^1,\ d_2 = e^2,\ d_3 = (u, v, 0),\ p_1 = (a, b, c),\ p_2 = (d, e, f)$ and $p_3 = \mathcal{O}$.

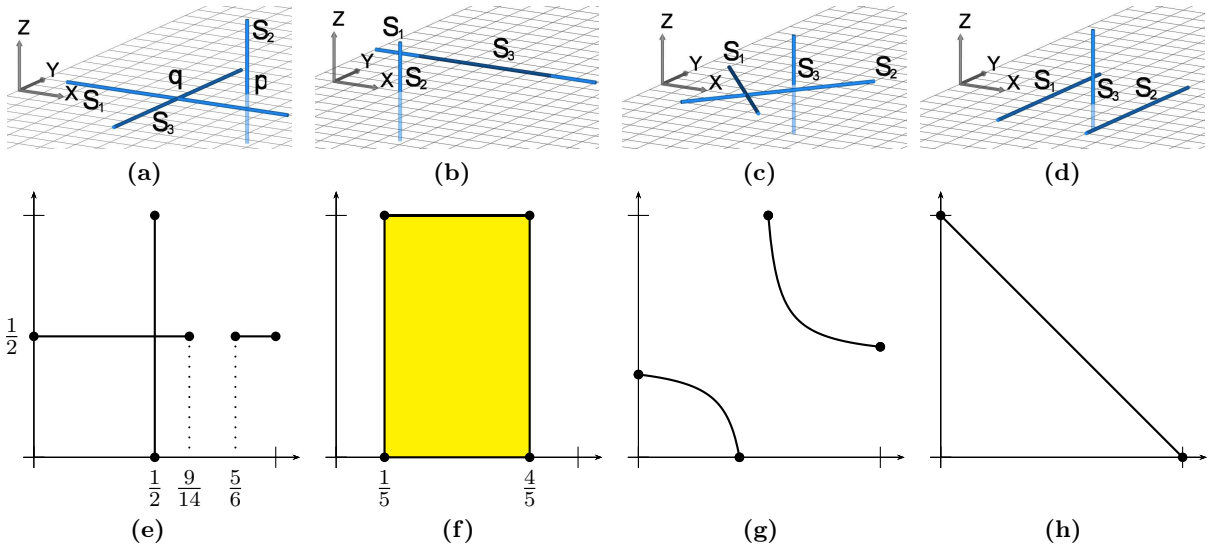


Figure 4: Mappings of three line segments, S_1, S_2, and S_3, in various configurations. The bottom figures depict the corresponding mapping $\Psi_{S_1S_2}(S_3)$. A point inside the yellow faces represents a line tangent to the three line segments. (a) S_1 and S_2 are skew, and S_1 and S_3 intersect at q. (e) $\Psi_{S_1S_2}(S_3)$ consists of two collinear horizontal line segments and one vertical line segment. (b) S_1 and S_2 are skew, and S_1 and S_3 overlap. (f) $\Psi_{S_1S_2}(S_3)$ is an axis parallel rectangle. (c) S_1 and S_2 are coplanar, and S_3 intersects the plane that contains S_1 and S_2 at a point. (g) $\Psi_{S_1S_2}(S_3)$ consists of two hyperbolic arcs. (d) S_1 and S_2 are parallel, and S_3 intersects the plane that contains S_1 and S_2 at a point. (h) $\Psi_{S_1S_2}(S_3)$ consists of a single line segment.

$\mathbf{c \neq 0}$, $\mathbf{f \neq 0}$ and $\mathbf{c \neq f}$: The three lines are skew. $\Psi_{L_1L_2}(L_3)$ is the line $t_2 = (-fvt_1 - cue + ufb + cdv - fav))/(cu)$. $\Psi_{L_1L_2}(S_3)$ is a segment defined between $t_1 = -a + cd/f$ and $t_1 = u - a + c(d-u)/f$. Note that this also covers the case $v = 0$ (L_3 parallel to L_1) for which the line becomes horizontal. For $u = 0$ (L_3 parallel to L_2) $\Psi_{L_1L_2}(S_3)$ is the vertical line segment $t_1 = -a + cd/f$, define between $t_2 = -e + bf/c$ and $t_2 = v - e + f(b-v)/c$.
$\mathbf{c \neq 0}$, $\mathbf{f \neq 0}$ and $\mathbf{c = f}$: L_1 and L_2 intersect at $p = (d, e, c) = (d, e, f)$. Since L_3 does not intersect the plane spanned by L_1 and L_2, $\Psi_{L_1L_2}(L_3)$ consists only of the point $(d-a, b-e)$ representing all lines containing p and L_3.
$\mathbf{c = 0}$, $\mathbf{f \neq 0}$ and $\mathbf{c \neq f}$:

- $\mathbf{v \neq 0}$: L_1 intersects L_3 at $p = (bu/v, b, 0)$. $\Psi_{L_1L_2}(S_3)$ is the vertical line $t_1 = -a + bu/v$ if $p \in S_3$ or empty otherwise.
- $\mathbf{v = 0 \wedge b \neq 0}$: L_1 is parallel to L_3 and since L_2 does not intersect the plane spanned by L_1 and L_2, it is obvious that $\Psi_{L_1L_2}(L_3) = \emptyset$.
- $\mathbf{v = 0 \wedge b = 0}$: S_3 overlaps with L_1; thus $\Psi_{L_1L_2}(S_3)$ is a two dimensional point set, namely, the vertical slab with t_1 between $-a$ and $u - a$.

This essentially also covers case $c \neq 0,\ f = 0$ for symmetry reasons.
$\mathbf{c = f = 0}$: L_3 is contained in the plane spanned by L_1 and L_2, thus $\Psi_{L_1L_2}(L_3) = \mathbb{R}^2 \setminus \ell$, where ℓ is the line representing those lines that are parallel to L_3. However, $\Psi_{L_1L_2}(S_3)$ is a bit more complex. $\Psi_{L_1L_2}(L_3(t_3))$ is the rectangular hyperbola with additional parameter t_3

$$t_2(t_1, t_3) = \frac{-((e - t_3 v)t_1 + (bu - eu + dv - av)t_3 + ea - db)}{(t_1 + a - t_3 u)}.$$

This family has two fix points that that do not depend on t_3: $p' = (d-a, b-e)$, which represents

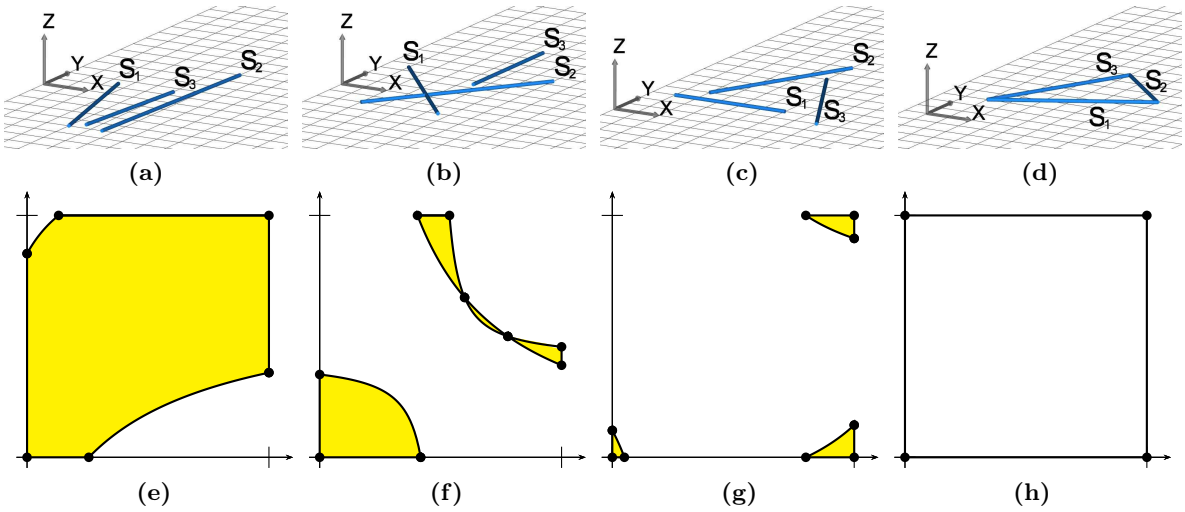


Figure 5: Mappings of three coplanar line segments, S_1, S_2, and S_3 in various configurations. The bottom figures depict the corresponding mapping $\Psi_{S_1S_2}(S_3)$. A point inside the yellow shape represents a line tangent to the three line segments. (a) S_1, S_2, and S_3 are pairwise disjoint. (b) S_1 and S_2 intersect, and S_3 is disjoint from both. (c) S_1, S_2, and S_3 are pairwise disjoint. (d) S_1, S_2, and S_3 form a triangle. (e) $\Psi_{S_1S_2}(S_3)$ consists of a single connected component. (f) $\Psi_{S_1S_2}(S_3)$ consists of four interior disjoint components. (g) $\Psi_{S_1S_2}(S_3)$ consists of three disconnected components. (h) $\Psi_{S_1S_2}(S_3)$ consists of the boundary of the unit square.

the intersection of L_1 and L_2; and $p'' = (-a + bu/v, -e + dv/u)$, which corresponds to the line L_3. Since two such hyperbolas can only intersect in at most two points we can conclude that the two-dimensional region $\Psi_{L_1L_2}(L_3([0,1] = S_3))$ is bounded by the hyperbolas $\Psi_{L_1L_2}(L_3(0))$ and $\Psi_{L_1L_2}(L_3(1))$; see also Figure 5. In the case $u = 0$ ($v = 0$) p'' is at infinity since all hyperbolas in the family have the same vertical (horizontal) asymptote.

A.2.2 Directions of L_1 and L_2 Are Dependent

We consider the case where L_1 and L_2 are linearly dependent. Thus, we can assume that $d_1 = (1,0,0)$, $d_2 = (u,0,0) \neq \mathcal{O}$, $d_3 = (0,1,0)$, $p_1 = (a,b,c)$, $p_2 = (d,e,f)$ and $p_3 = \mathcal{O}$.
$\mathbf{c \neq 0}$, $\mathbf{f \neq 0}$: L_1 and L_2 are parallel and do not intersect L_3. L_3 intersects the plane spanned by L_1 and L_2 in $p = (0, (ce - fb)/(c - f), 0)$. $\Psi_{L_1L_2}(L_3)$ is the line $t_2 = (fa + ft_1 - cd)/(cu)$. $\Psi_{L_1L_2}(S_3) = \emptyset$ iff $p \notin S_3$.
$\mathbf{c = 0}$, $\mathbf{f \neq 0}$: L_1 and L_2 are parallel. L_1 intersects L_3 at $p = (0,b,0)$. $\Psi_{L_1L_2}(L_3)$ is the line vertical line $t_1 = -a$. $\Psi_{L_1L_2}(S_3) = \emptyset$ iff $p \notin S_3$.
$\mathbf{c \neq 0}$, $\mathbf{f = 0}$: L_1 and L_2 are parallel. L_2 intersects L_3 at $p = (0,e,0)$. $\Psi_{L_1L_2}(L_3)$ is the horizontal line $t_2 = -d/u$. $\Psi_{L_1L_2}(S_3) = \emptyset$ iff $p \notin S_3$.
$\mathbf{c = 0}$, $\mathbf{f = 0}$: L_1 and L_2 are parallel and intersect L_3 at $p' = (0,b,0)$ and $p'' = (0,e,0)$. Thus $\Psi_{L_1L_2}(L_3) = \mathbb{R}^2 \setminus \ell$, where ℓ is the line representing those lines that are parallel to L_3. This case is similar to the one in Subsection A.2.1. $\Psi_{L_1L_2}(L_3(t_3))$ is a line $u(b - t_3)t_2 = (-at_3 - t_1t_3 + dt_3 - db + et_1 + ea)$. The family has a fix point $p = (-a, -d/u)$, which corresponds to the line L_3. $\Psi_{L_1L_2}(L_3([0,1]))$ is a wedge which is bounded by the lines $\Psi_{L_1L_2}(L_3(0))$ and $\Psi_{L_1L_2}(L_3(1))$.

B The Complete Algorithm

We describe the complete algorithm that handles all degenerate cases, in particular, line segments that degenerate to points, concurrent line segments, and collinear, including overlapping, line segments.

We set the line segments that degenerate to points apart from the rest. Let $\mathcal{S}'$ denote the subset of such degenerate line segments, and $\mathcal{S}''$ denote the full-dimensional line segments, $\mathcal{S} = \mathcal{S}' \cup \mathcal{S}''$. The entire algorithm consists of three phases. During the first phase we go over all points in $\mathcal{S}'$. For each point S_i we find every line that contains S_i, possibly contains other points in $\mathcal{S}'$, and possibly intersects line segments in $\mathcal{S}''$, such that the total number of imposing constraints is at least four. During the second phase, we go over pairs of the full-dimensional line segments in $\mathcal{S}''$ that are not collinear. For each pair, (S_i, S_j), we find lines that intersect S_i and S_j and other line segments in $\mathcal{S}''$, such that the total number of imposing constraints is at least four. These constraints, however, might be imposed by collinear input line segments. The processing of groups of four or more collinear line segments is deferred to the third phase. Such groups can easily be detected by lexicographically sorting the line segments by their normalized Plücker coordinates. W. l. o. g. assume that $\mathcal{S}$ consists of $\mathcal{S}''$ sorted accordingly preceded by $\mathcal{S}'$. Also, assume that $\mathcal{S}'$ consists of m points; see Algorithm 3 for pseudo code (see Section 3 for definitions).

Algorithm 3 Compute lines that intersect line segments in $\mathcal{S} = \{S_1, \ldots, S_n\}$.

Phase I

1.1 **for** $i = 1, \ldots, m$,
1.2.1 Construct the arrangement $\mathcal{A}^s_{S_i}$ induced by $\{\Xi_{S_i}(S_k) \,|\, k = i+1, \ldots, n\}$.
1.2.2 Extract lines that intersect S_i from $\mathcal{A}^s_{S_i}$.

Phase II

2.1 **for** $i = m+1, \ldots, n-3$,
2.2 **for** $j = n, \ldots, i+3$,
2.3 **if** S_i and S_j are collinear, **break**.
2.5.1 Construct the arrangement $\mathcal{A}_{S_iS_j}$ induced by $\{\Psi_{S_iS_j}(S_k) \,|\, k = i+1, \ldots, j-1\}$.
2.5.2 Extract lines that intersect S_i and S_j from $\mathcal{A}_{S_iS_j}$.
2.6 **if** S_i and S_j intersect,
2.7.1 Construct the arrangement $\mathcal{A}^s_{S_i \cap S_j}$ induced by $\{\Xi_{S_i \cap S_j}(S_k) \,|\, k = i+1, \ldots, j-1\}$.
2.7.2 Extract lines that intersect S_i and S_j from $\mathcal{A}^s_{S_i \cap S_j}$.

Phase III

3.1 $i \leftarrow m+1$.
3.2 **while** $i \leq n-3$,
3.3 $j \leftarrow i+1$.
3.4 **while** $j \leq n$,
3.5 **if** S_i and S_j are not collinear,
3.6 **if** $i+3 < j$,
3.7 Process the set of collinear line segments $S_i, \ldots, S_{j-1}$.
3.8 $i \leftarrow j$.
3.9 **break**.
3.10 $j \leftarrow j+1$.
3.11 **if** $i+3 < j$,
3.12 Process the set of collinear line segments $S_i, \ldots, S_{j-1}$.
3.13 $i \leftarrow j$.

We defer the description of Phase I to Appendix B.2, and proceed with the description of Phase II.

B.1 The Processing of Arrangements in the Plane

B.1.1 Constructing the Arrangement in the Plane

In Line 2.5.1 of Algorithm 3 we construct the arrangement $\mathcal{A}_{S_iS_j}$ induced by the set $\mathcal{C}_{ij} = \{\Psi_{S_iS_j}(S_k) \,|\, k = i+1, \ldots, j-1\}$. Notice that S_i and S_j are not collinear. We distinguish between two disjoint subsets $\mathcal{C}'_{ij}$ and $\mathcal{C}''_{ij}$ of $\mathcal{C}_{ij}$, such that $\mathcal{C}'_{ij}$ consists of one-dimensional hyperbolic arcs restricted to the unit square, see Appendix A, and $\mathcal{C}''_{ij}$ consists of two-dimensional regions bounded by hyperbolic arcs and the unit square. (Recall that $\Psi_{S_iS_j}(S_k)$ consists of two-dimensional regions if either S_i or S_j overlap with S_k, or S_i, S_j, and S_k are coplanar; see Appendix A.) We process the line segments $S_{i+1}, \ldots, S_{j-1}$ one at a time to produce the inducing point sets $\mathcal{C}'_{ij}$ and $\mathcal{C}''_{ij}$. Next, using a plane-sweep algorithm, we construct the arrangement $\mathcal{A}'_{S_iS_j}$ induced by $\mathcal{C}'_{ij}$. The processing of $\mathcal{C}''_{ij}$ is different, because $\mathcal{C}''_{ij}$ consists of two-dimensional regions. For each maximal two-dimensional point set $\mathcal{R}_k \in \mathcal{C}''_{ij}$, $\mathcal{R}_k \subseteq \Psi_{S_iS_j}(S_k)$, we construct the arrangement $\mathcal{A}^k_{S_iS_j}$ induced by $\mathcal{R}_k$. Then, we construct $\mathcal{A}''_{S_iS_j}$ by overlaying all the arrangements in $\{\mathcal{A}^k_{S_iS_j} \,|\, \mathcal{R}_k \in \mathcal{C}''_{ij}\}$. Finally, we overlay the arrangements $\mathcal{A}'_{S_iS_j}$ and $\mathcal{A}''_{S_iS_j}$ to produce the final arrangement $\mathcal{A}_{S_iS_j}$; see Figure 6. The *2D Arrangements* package of CGAL supports an overlay operation, which computes the overlay of two given arrangements. In practice, we apply this operation several times to compute $\mathcal{A}''_{S_iS_j}$.

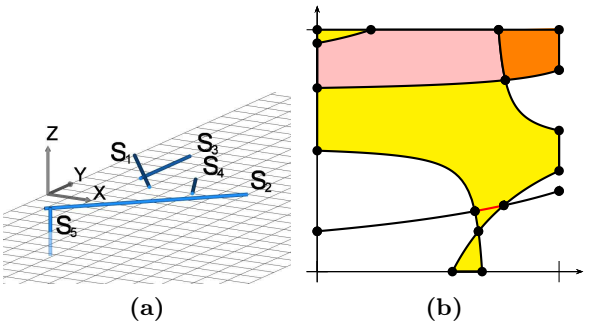


Figure 6: (a) Five line segments, $S_1, \ldots, S_5$, such that the first four are coplanar, and S_5 pierces the plane containing the first four. (b) The arrangement $A_{S_1S_2}$. The pink face represents a ruled surface patch the lines of which intersect the four coplanar line segments. The red curve represents a ruled surface patch the lines of which intersect S_1, S_2, S_3, and S_5.

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. Similarly, we store with each face of the arrangement $\mathcal{A}''_{S_iS_j}$ the sorted sequence of the line segments mapped through $\Psi_{S_iS_j}$ to the regions that induce that face. Each cell, i.e., vertex, edge, or face, of $\mathcal{A}_{S_iS_j}$ stores the sequence of the line segments obtained during the overlay from the corresponding cell in $\mathcal{A}'_{S_iS_j}$ and $\mathcal{A}''_{S_iS_j}$. We store only the minimal necessary set of line segments in every sequence. A member of a sequence of a vertex is a line segment that maps to a (zero-dimensional) point $p \in \mathcal{C}'_{ij}$. A member of a sequence of an edge is a line segment that maps to a (one-dimensional) curve $C \in \mathcal{C}'_{ij}$. A member of a sequence of a face is a line segment that maps to a (two-dimensional) region $R \in \mathcal{C}''_{ij}$. For example, consider a curve $C \in \mathcal{C}'_{ij}$, such that $C \subseteq \Psi_{S_iS_j}(S)$. The sequences of all edges induced by C contain S. However, the sequences of line segments of all vertices incident to these edges do not contain S, as this information is immediately accessible from the incident edges. Similarly, the sequences of all faces induced by a region $R \subseteq \Psi_{S_iS_j}(S)$ contain the line segment S, but the sequences of all vertices and edges incident to these faces do not. When a new face, f, is formed while overlaying two arrangements, say $\mathcal{A}_1$ and $\mathcal{A}_2$, we merge the sequence of line segments of the two faces, f_1 and f_2 of $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively, that induce f, and store the resulting sequence in f.

B.1.2 Extracting the Information and User Options

In Line 2.5.2 of Algorithm 3 we extract the information from the arrangement $\mathcal{A}_{S_iS_j}$ and provide it to the user in a usable format. Recall that a vertex represents a single line, while an edge or a face represent a surface patch. A line or a surface patch are provided as output only if it abides by four intersection constraints. Recall that the user can choose whether to count overlapping segments as one constraint (the default), or as many constraints. If the user chooses to count overlapping segments as one constraint, we ignore overlapping line segments when we traverse the sequences of line segments stored with the arrangement cells. As the line segments are sorted by their normalized Plücker coordinates, overlapping line segments are adjacent, and thus easy to detect. Observe that two different line segments that map to the same hyperbolic arc may not necessarily overlap; see Figure 3b. We provide the user with the ability to iterate over eligible cells of different dimensions separately. By default we consider a surface patch of the output represented by an edge or a face open. The user can override this setting, and choose to consider surface patches closed. In this case the iteration over the zero-dimensional output elements results with only eligible vertices that are not incident to eligible edges. Similarly, the iteration over the one-dimensional output elements results with only eligible edges that are not incident to eligible faces.

By default, the generating line segments of every output surface patch or line are provided as part of the output. For example, if a line, L, intersects five line segments, $S_1, \ldots, S_5$, in general position, then L appears in five different output elements, each with a different subset of four generating line segments. The user can override this default setting and choose (i) not to include the generating line segments in the output, and (ii) to obtain each output line or surface patch exactly once. When the user chooses to exclude the generating line segments, we revert the default setting of the second option to produce a compact output set that does not include repetitions; that is, a given line is not returned more than once (as a single line or as part of a ruled surface patch). Providing a compact output requires slightly more processing, which however does not compromise the asymptotic complexity of the algorithm. The flexibility we allow adds some technical and conceptual difficulties as explained below. However, it makes the implementation practical for real-life applications.

Consider the line L above and assume that the user chooses to obtain a compact output set. L must be provided only once as output. However, executing Algorithm 3 results with repetitions; the line L is provided twice, because two vertices, say v_{15} and v_{14}, that represent the line L exist in two different arrangements, $A_{S_1S_5}$ and $A_{S_1S_4}$, respectively. The sequence of line segments stored in v_{15} consists of S_2, S_3, S_4, and, thus, qualifies v_{15} as an eligible vertex. Similarly, the sequence of line segments stored in v_{14} consists of S_2, S_3, and qualifies v_{14} as an eligible vertex as well. To avoid repetitions, we alter the algorithm as follows: We consider the augmented set $\{\Psi_{S_iS_j}(S_k) \,|\, k = 1, \ldots, i{-}1, i{+}1, \ldots, j{-}1, j{+}1, \ldots, n\}$ as the inducing set of the arrangement $\mathcal{A}_{S_iS_j}$ in Line 2.5.1. At first glance this change may seem to produce even a larger amount of repetitions. Considering our example, there are $\binom{5}{2}$ different arrangements that contain an eligible vertex each that represents the line L. We alter the iterations over the eligible cells to filter out cells that fail an additional criterion. Let c be an eligible cell in the arrangement $\mathcal{A}_{S_iS_j}$ and let $S_{a_1}, \ldots, S_{a_\ell}$ be the generating line segments stored in c. We output the line or surface patch represented by c, only if $i < a_1, \ldots, a_\ell < j$. In our example this would be the vertex v_{15} of the arrangement $\mathcal{A}_{S_1S_5}$. Observe that every line that abides by at least four intersection constraints is represented by several vertices of different arrangement due to the augmentation of the inducing point sets. An alternative method to avoid repetitions of lines in the output, for example, is to insert them first into a set data structure using their Plücker coordinates as keys and filter out repetitions.

B.2 The Processing of Arrangements on the Sphere

B.2.1 Constructing the Arrangement

In Line 2.7.1 of Algorithm 3 we construct an arrangement on the sphere centered at p, the point of intersection between S_i and S_j. The arrangement is induced by the point set $\mathcal{C}^s_{ij} = \{\Xi_{S_i \cap S_j}(S_k) \,|\, k = i+1, \ldots, j-1\}$. We maintain a set, $\mathcal{Q}_{ij}$, of all the line segments $S_k, i < k < j$ that contain p. We process the line segments $S_{i+1}, \ldots, S_{j-1}$ one at a time to produce the inducing set $\mathcal{C}^s_{ij}$. If S_k contains the sphere center p, we insert S_k into $\mathcal{Q}_{ij}$ and proceed. 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 arc on $\mathbb{H}^2$; see Section 2.2. These arcs or point represent lines that intersect S_i and S_j at p and also intersect S_k. The intersections of points and arcs in $\mathcal{C}^s_{ij}$ represent lines that intersect at least four input segments. Next, using a sweep of the sphere algorithm, 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.

Similarly, in Line 1.2.1 of Algorithm 3, we construct the arrangement $\mathcal{A}^s_{S_i}$ on the sphere centered at S_i to account for every line that contains the point S_i and abides by at least two or three additional constraints (depending on the user settings). The arrangement is induced by the sets $\mathcal{C}'^s_i$ and $\mathcal{C}''^s_i$. We process the points $S_{i+1}, \ldots, S_m$ one at a time to produce the inducing set $\mathcal{C}'^s_i = \{\Xi_{S_i}(S_k) \,|\, k = i+1, \ldots, m\}$, which consists of points. We process the (full-dimensional) line segments $S_{m+1}, \ldots, S_n$ one at a time to produce the inducing set $\mathcal{C}''^s_i = \{\Xi_{S_i}(S_k) \,|\, k = i+1, \ldots, m\}$. As in the case above, we maintain a set, $\mathcal{Q}_{ij}$, of all the line segments from $S_{m+1}, \ldots, S_n$ that contain the point S_i. If S_k contains the point S_i, we insert S_k into $\mathcal{Q}_{ij}$ and proceed. When the underlying line of a line segment S_k contains the sphere center, $\Xi_{S_i}(S_k)$ consists of a single point. Next, using a sweep of the sphere algorithm, we construct the arrangement $\mathcal{A}^s_{S_i}$ induced by $\mathcal{C}^s_i = \mathcal{C}'^s_i \cup \mathcal{C}''^s_i$.

In both cases we extend the vertex and edge records. In the former case 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 the latter case we store with each vertex and edge of the arrangement $\mathcal{A}^s_{S_i}$ the sorted sequence of input (zero and full-dimensional) line segments that are mapped through Ξ_{S_i} to the points and geodesic arcs that induce that cell.

B.2.2 Extracting the Information and User Options

In Line 1.2.2 and Line 2.7.2 of Algorithm 3 we extract the information from the arrangements on the sphere and provide it to the user in a usable format. All the settings that apply to the processing of the arrangements in the plane apply to the processing of the arrangements on the sphere as well; see Section B.1.2. Recall that by default an element of the output is a pair of a line or a surface patch and a tuple of generating line segments that impose exactly four constraints. If the user does not change this default setting, the set, $\mathcal{Q}_{ij}$, of line segments that contain the intersection point p is ignored. Otherwise, all the line segments in this set are considered generating segments. As with the aforementioned processing of the arrangements in the plane, we provide the user with the ability to iterate over eligible vertices and over eligible edges separately. In addition, we allow the user to exclude the generating segments from the output, and, independently, to obtain an output set without repetitions. When the user selects such a compact output, we alter the processing of the arrangement in the sphere in a similar way we alter the processing of the arrangement in the plane. We consider the augmented set $\{\Xi_{S_i \cap S_j}(S_k) \,|\, k = 1, \ldots, i-1, i+1, \ldots, j-1, j+1, \ldots, n\}$ as the inducing set of the arrangement $\mathcal{A}^S_{S_i S_j}$ in Line 2.7.1, and filter out cells that fail the following criterion: Let c be an eligible cell in the arrangement $\mathcal{A}^s_{S_i S_j}$ and let $S_{a_1}, \ldots, S_{a_\ell}$ be the generating line segments stored in c. We output the line or surface patch represented by c, only if $i < a_1, \ldots, a_\ell < j$.

B.3 The Processing of Collinear Line Segments

Let $\mathcal{L} = \{S_1, ..., S_\ell\}$ be a set of $4 \leq \ell$ collinear line segments and let L be their underlying line. Assume $\mathcal{L}$ is processed in Line 3.7 or Line 3.12. If the user does not alter the default settings, we generate $\binom{\ell}{4}$ output elements. Every output element consists of the line L and a different quadruple of line segments from $\mathcal{L}$. If the user chooses a compact output, L becomes the sole candidate for the output. We return L only if it does not intersect any one of the line segments in $\mathcal{S} \setminus \mathcal{L}$ to avoid repetitions.

If the user alters the default settings, and chooses to consider every input line segment as a single constraint, we also need to account for all lines that intersect sub-segments that are the intersections of at least four input line segments. We construct a one-dimensional arrangement $\mathcal{A}^\ell$ embedded in the line L induced by the line segments in $\mathcal{L}$. We store with each vertex and edge of $\mathcal{A}^\ell$ the inducing line segments of that cell. As with the aforementioned processing of two-dimensional arrangements, we provide the user with the ability to iterate over eligible vertices and over eligible edges separately. As in the case above, if the user chooses a compact output, we return a line only if it does not intersect any one of the line segments in $\mathcal{S} \setminus \mathcal{L}$. Observe that in this case an eligible edge, the geometric embedding of which is the overlapping sub segment of some input line segments, represents a ruled surface patch. We split this surface patch into as many as necessary lines and surface patches that do not intersect any one of the line segments in $\mathcal{S} \setminus \mathcal{L}$.

B.4 Complexity Analysis

Setting apart the subset $\mathcal{S}'$ of line segments that degenerate to points is naturally done in linear time. Sorting the subset, $\mathcal{S}''$, of full-dimensional line segments according to their normalized Plücker coordinates is done in $O(n \log n)$ time. Constructing the arrangement $\mathcal{A}^s_{S_i}$ for a given point $S_i \in \mathcal{S}'$ is done in $O((n+k_i) \log n)$ time using a plane-sweep algorithm, where k_i in the number of intersections of the inducing geodesic arcs. The total time it takes to construct all arrangements $\{\mathcal{A}^s_{S_i} \,|\, i = 1, \ldots, m\}$ is $O((mn + I_1) \log n)$, where I_1 is the total number of respective intersections. Extracting the output lines from all arrangements $\{\mathcal{A}^s_{S_i} \,|\, i = 1, \ldots, m\}$ takes $O((mn + I_1))$ time in total. Given two line segments $S_i, S_j \in \mathcal{S}''$, constructing the arrangement $\mathcal{A}'_{S_iS_j}$ is done in $O((n+k'_{ij}) \log n)$ time using a plane-sweep algorithm, where k'_{ij} in the number of intersections of the inducing hyperbolic arcs. Constructing the arrangement $\mathcal{A}''_{S_iS_j}$ can also be done in $O((n + k''_{ij}) \log n)$, where k''_{ij} in the number of intersections of the inducing regions. Computing the overlay of $\mathcal{A}'_{S_iS_j}$ and $\mathcal{A}''_{S_iS_j}$ is done in $O((n + k_{ij}) \log n)$ time, where k_{ij} in the number of intersections of the curves of the two arrangements. If S_i and S_j intersect, we also construct the arrangement $\mathcal{A}^S_{S_iS_j}$ in $O((n + k^s_{ij}) \log n)$ time, where k^s_{ij} in the number of intersections of the geodesic arcs. Thus, the total time it takes to construct and process all arrangements in Phase II is $O((n^3 + I_2) \log n)$, where I_2 is the total number of respective intersections. The asymptotic resource-consumption of Phase III is negligible compared to those of the other phases. In summary, 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.

B.5 Output Quality

The output of Algorithm 3 is canonical, because the input line segments are sorted by their normalized Plücker coordinates to start with. However, the output is not canonical, for example, when the input is subject to affine transformation. Consider the segments in $\mathcal{S} = \{S_1, S_2, S_3, S_4\}$ de-

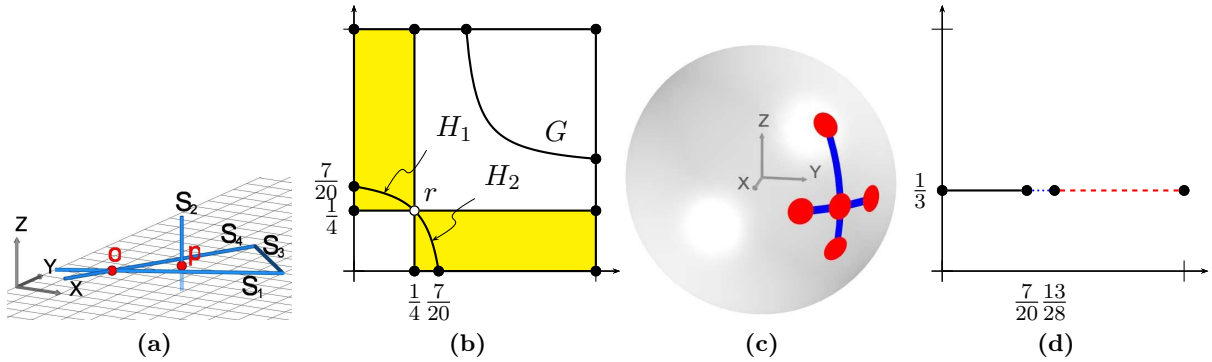


Figure 7: The processing of three coplanar line segments. (a) S_1, S_2, and S_3 are coplanar. S_4 pierces the plane containing S_1, S_2, and S_3. (b) The arrangement $\mathcal{A}_{S_1S_4}$ induced by $\Psi_{S_1S_4}(S_2 \cup S_3)$. (c) The arrangement $\mathcal{A}^s_{S_1 \cap S_4}$ induced by $\Xi_{S_1 \cap S_4}(S_2 \cup S_3)$. (d) The arrangement $\mathcal{A}_{S_1S_2}$ induced by $\Psi_{S_1S_2}(S_3 \cup S_4)$.

picted in Figure 7a. The segments S_1, S_3, and S_4 are coplanar; they all lie in the plane P. The forth segment, S_2, intersects P at the point p. The point set $\Psi_{S_1S_4}(S_3)$, depicted in Figure 7b, consists of two disconnected axes-aligned rectangles that in particular do not include the point r, as it does not represent a valid line. The point set $\Psi_{S_1S_4}(S_2)$, depicted in the figure as well, consists of three hyperbolic arcs, H_1, H_2, and G, where H_1 and H_2 lie on the same hyperbola and split at r. Feeding $\mathcal{S}$ as input to Algorithm 3 results with five output elements: the line L_1 containing p and the intersection point $S_1 \cap S_3$, the line L_2 containing p and the intersection point $S_4 \cap S_3$, two surface patches represented by the two hyperbolic arcs H_1 and H_2, respectively, and the line, L_3, containing p and the intersection point $S_1 \cap S_4$. The latter results from the processing of the arrangement $\mathcal{A}^s_{S_1 \cap S_4}$ on the sphere; see Figure 7c. On the other hand, feeding $\mathcal{S}' = \{S_1, S_3, S_4, S_2\}$ as input to Algorithm 3 results with three output elements; see Figure 7d. An absolute canonical output might be hard to come up with. However, we do allow the user to select a criterion for the output set trading off different characteristics as explained next. First, we go over all unordered pairs (i, j); that is, Line 2.1 and Line 2.2 of Algorithm 3 become:

2.1 **for** $i = m + 1, \ldots, n - 1$,
2.2 **for** $j = n, \ldots, i + 1$,

Next, we consider the augmented set $\{\Psi_{S_iS_j}(S_k) \,|\, k = 1, \ldots, i-1, i+1, \ldots, j-1, j+1, \ldots, n\}$ as the inducing set of the arrangement $\mathcal{A}_{S_iS_j}$ in Line 2.5.1 and the augmented set $\{\Xi_{S_i \cap S_j}(S_k) \,|\, k = 1, \ldots, i-1, i+1, \ldots, j-1, j+1, \ldots, n\}$ as the inducing set of the arrangement $\mathcal{A}^S_{S_iS_j}$ in Line 2.7.1—the same augmented sets we consider to avoid repetitions; see Section B.1.2 and Section B.2.2. Finally, we change the criterion that we use to filter out ineligible cells as follows. Let c be an eligible cell in the arrangement $\mathcal{A}_{S_iS_j}$ or $\mathcal{A}^s_{S_i \cap S_j}$. We output the line or surface patch represented by c, only if c satisfies the user selected criterion. We support two criteria: one that favors the planar arrangements and another that favors the spherical arrangements. If the user chooses the former criterion, we do not use arrangements on the sphere at all. Instead, we extract the output from the planar arrangements. In this case we obtain the minimal number of output elements, which are also maximally connected. If the user chooses the latter criterion, we extract as many as output elements as possible from the spherical arrangements.