

An Affine-invariant Time-dependent Triangulation of Spatio-temporal Data*

Sofie Haesevoets and Bart Kuijpers[†]
Hasselt University, Belgium

Abstract

In the *geometric data model* for spatio-temporal data, introduced by Chomicki and Revesz [6], spatio-temporal data are modelled as a finite collection of triangles that are transformed by time-dependent affinities of the plane. To facilitate querying and animation of spatio-temporal data, we present a *normal form* for data in the geometric data model. We propose an algorithm for constructing this normal form via a *spatio-temporal triangulation* of geometric data objects. This triangulation algorithm generates new geometric data objects that partition the given objects both in space and in time. A particular property of the proposed partition is that it is *invariant under time-dependent affine transformations*, and hence independent of the particular choice of coordinate system used to describe the spatio-temporal data in. We can show that our algorithm works correctly and has a polynomial time complexity (of reasonably low degree in the number of input triangles and the maximal degree of the polynomial functions that describe the transformation functions). We also discuss several possible applications of this spatio-temporal triangulation.

1 Introduction and Summary

At this moment, spatial databases are a well-established area of research. Since most natural and man-made phenomena have a temporal as well as a spatial extent a lot of attention has also been paid, during the last decade, to modelling and querying spatio-temporal data [3, 4, 5, 15, 17, 18, 24, 29]. Several data models for representing spatio-temporal data have been proposed already. In this article, we adopt the *geometric data model* that was

*An extended abstract of an earlier version of this paper appeared as [20].

[†]Contact author: bart.kuijpers@uhasselt.be

introduced by Chomicki and Revesz [6] and of which closure properties under boolean operations were later studied by them and the present authors [5, 23]. In the geometric data model, *spatio-temporal objects* are finitely represented as *geometric objects*, which in turn are collections of atomic geometric objects. An atomic geometric object is given by its spatial reference object (which determines its shape), a time interval (which specifies its lifespan) and a transformation function (which determines the movement of the object during the time interval).

Although this model is very natural, it is not immediately clear how the spatio-temporal object represented by some atomic geometric objects looks like. This difficulty has several possible reasons. To start with, the time domain of the spatio-temporal object has to be computed from the time domains of all atomic geometric objects, which might overlap or may contain gaps (i.e., moments when the spatio-temporal object does not exist). Furthermore, two different sets of atomic geometric objects may represent the same spatio-temporal object and there may be elements in the set of atomic geometric objects that do not contribute to the spatio-temporal object at all, as they may be overlapped totally by other atomic objects. In short, the proposed geometric data model would benefit from a *normal form* that supports visualization and describes the objects in a unique way.

We propose as a normal form an *affine-invariant spatio-temporal triangulation*. This triangulation can be used to preprocess geometric objects in order to facilitate querying and animation in such a way that queries can be executed much more efficiently and require few additional computations. The main reason for this is that in a spatio-temporal triangulation the data is partitioned in space as well as in time. Hence, no objects overlap, thus reducing unnecessary computations. Actually, we deviate a little from the strict mathematical concept of a triangulation and allow triangles in a spatial and spatio-temporal triangulation to share boundaries with each other, as is not uncommon (see, e.g., [14]).

Our spatio-temporal triangulation is also *invariant under time-dependent affinities*. In the area of spatial database research, much attention has been paid to affine invariance of both data description and manipulation techniques and queries [16, 19, 27]. The main idea of working in an affine invariant way is to obtain methods and techniques that are not affected by affine transformations of the ambient space in which the data is situated. This means that a particular choice of origin or some particular, possibly artificial, choice of unit of measure (e.g., inches, centimeters, ...) and direction of coordinate axis has no effect on the final result of the method, technique or query. This means that an affine-invariant method is robust with respect to a particular choice of measuring data.

Also in other areas, invariance under affinities is often relevant. In computer vision, the so-called *weak perspective assumption* [31] is widely adopted.

	Time Complexity	Output complexity
Spatial data	$O(m^2 \log m)$	$O(m^2)$
Spatio-temporal data	$O(z(d, \epsilon)dm^5 \log m)$	$O(m^5d)$

Table 1: Summary of the complexity results.

This assumption says that when an object is repeatedly photographed from different viewpoints, and the object is relatively far away from the camera, that all pictures of the object are affine images of each other, i.e., all images are equal up to an affinity of the photographic plane. We generalize this assumption for spatio-temporal objects as follows. If a spatio-temporal event is filmed by two moving observers, relatively far away from the event, then both films will be the same up to a time-dependent affinity of the plane of the pellicle. For each time moment, another affinity maps the snapshots of the different movies onto each other.

The weak perspective assumption has necessitated affine-invariant similarity measures between pairs of pictures [21, 22, 32]. Also, in computer graphics, affine-invariant norms and triangulations have been studied [25]. In the field of spatial and spatio-temporal constraint databases [28, 30], affine-invariant query languages [16, 19, 27] have been proposed. For spatial data, there exist several triangulation algorithms, but, apart from the triangulation of Nielson [25], their output is not affine-invariant. The method proposed by Nielson to triangulate a set of points in an affine-invariant way computes an affine-invariant norm using the coordinate information of all points, and then uses this norm in the triangulation algorithm. We develop a spatial triangulation algorithm that is more intuitive, that is efficiently computable and that naturally extends to a spatio-temporal triangulation algorithm.

The main contribution of this paper is an affine-invariant time-dependent triangulation algorithm that produces a unique and affine-invariant triangulation of a spatio-temporal object given as a geometric data object. As mentioned before, a geometric input object for this triangulation algorithm, consists of m atomic geometric objects, given by a triangle, a time interval and a time-dependent transformation function that we assume to be given as a fraction of polynomial functions of degree at most d . We show that our triangulation algorithm runs in polynomial time in the size of the input, measured by m and d . The worst-case time complexity is of order $z(d, \epsilon)dm^5 \log m$, where $z(d, \epsilon)$ is the complexity of finding all roots of an univariate polynomial of degree d , with accuracy ϵ . The maximal number of atomic objects in the resulting triangulation is of order m^5d^6 . For static spatial data the time complexity of triangulating is of order $m^2 \log m$ and the number of returned triangles is of order m^2 . These results are summarized in Table 1.

We remark that such triangulations could also be computed via general

purpose cell decomposition algorithms, most notably cylindrical algebraic decomposition [7]. These algorithms are not affine-invariant, however, and are therefore not directly suitable for the computational task that we consider.

In this paper, we also show some applications of the proposed triangulation and show that, when computed in a preprocessing stage, it facilitates the computation of certain types of queries and operations.

The outline of this paper is as follows. In Section 2, we explain the geometric data model and define spatial and spatio-temporal triangulations. We introduce an affine invariant spatial triangulation method in Section 3. Afterwards, in Section 4, we describe a novel affine-invariant triangulation of spatio-temporal data. We describe the algorithm in detail and give and prove some properties. Then, we give some possible applications of the triangulation in Section 5 and we end with some concluding remarks in Section 6.

2 Preliminaries and Definitions

We denote the set of real numbers by \mathbb{R} and the two-dimensional real space by \mathbb{R}^2 . The space containing moving 2-dimensional objects will be denoted $(\mathbb{R}^2 \times \mathbb{R})$. We will use x and y (with or without subscripts) to denote spatial variables and t (with or without subscripts) to denote time variables. The letter T (with or without subscripts) will be used to refer to triangles, which we assume to be represented by triples of pairs of points in \mathbb{R}^2 .

In this section, we first give the definition of a spatial, a temporal and a spatio-temporal object. Next, we come back to the need of a normal form. Finally, we define affine triangulations of spatial and of spatio-temporal data.

2.1 Spatio-temporal Data in the Geometric Data Model

In this section, we describe the geometric data model as introduced by Chomicki and Revesz [6], in which spatio-temporal data are modeled by geometric objects that in turn are finite collections of atomic (geometric) objects. First, we define temporal, spatial and spatio-temporal data objects. In this definition we work with semi-algebraic sets because these are infinite sets that allow a effective finite description by means of polynomial equalities and inequalities. More formally, a semi-algebraic set in \mathbb{R}^d is a Boolean combination of sets of the form $\{(x_1, x_2, \dots, x_d) \in \mathbb{R}^d \mid p(x_1, x_2, \dots, x_d) > 0\}$, where p is a polynomial with integer coefficients in the real variables x_1, x_2, \dots, x_d . Properties of semi-algebraic sets are well known [2].

Definition 1 A *temporal object* is a semi-algebraic subset of \mathbb{R} , a *spatial object* is a semi-algebraic subset of \mathbb{R}^2 and a *spatio-temporal object* is a semi-algebraic subset of $(\mathbb{R}^2 \times \mathbb{R})$. \square

With the *time domain* of a spatio-temporal object, we mean its projection on the time axis, i.e., on the third coordinate of $(\mathbb{R}^2 \times \mathbb{R})$. It is a well-known property of semi-algebraic sets, that this projection is a semi-algebraic set and can therefore be considered a temporal object [2].

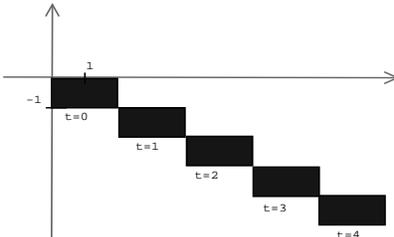


Figure 1: An example of a spatio-temporal object.

Example 1 The interval $[0, 4]$ and the finite set $\{0, 1, 2, 3, 4\}$ are examples of temporal objects. The unit circle in the plane is a spatial object, since it can be represented by the polynomial inequalities $\neg(1 - x^2 - y^2 > 0) \wedge \neg(x^2 + y^2 - 1 > 0)$, usually abbreviated by the formula $x^2 + y^2 = 1$. The set $\{(x, y; t) \in \mathbb{R}^2 \times \mathbb{R} \mid x \geq 2t \wedge x \leq 2 + 2t \wedge y \geq -t - 1 \wedge y \leq -t \wedge t \geq 0 \wedge t \leq 4\}$ is a spatio-temporal object and it represents rectangle that is translated at constant speed during the time interval $[0, 4]$. At each moment t in this interval it has corner points $(2t, -t)$, $(2 + 2t, -t)$, $(2 + 2t, -t - 1)$ and $(2t, -t - 1)$, as illustrated in Figure 1. \square

In *the geometric data model* [5, 6], spatio-temporal objects are finitely represented by geometric objects, which in turn are finite collections of atomic geometric objects. An atomic geometric object is given by its spatial reference object (which determines its shape), a time interval (which specifies its lifetime) and a transformation function (which determines the movement of the object during the time interval).

Several classes of geometric objects were introduced, depending on the types of spatial reference objects and transformations [5, 6]. In this article, we consider *spatio-temporal objects that can be represented as finite unions of triangles moved by time-dependent affine transformations* (see also Definition 2). Other combinations have been studied [5] in which triangles, rectangles or polygons are transformed by time-dependent translations, scalings or affinities, that, in turn, are given by linear, polynomial or rational functions of time. The class of geometric objects that we consider is not only the most general of the classes that were previously studied, it is also one of the few classes that have the desirable property of being closed under the set operations union, intersection and difference [5]. In Section 4, we will rely on this closure property.

Definition 2 An *atomic geometric object* \mathcal{O} is a triple (T, I, f) , where

- $T \subset \mathbb{R}^2$ is the *spatial reference object* of \mathcal{O} , which is a (filled) triangle with corner points that have rational or algebraic coefficients¹;
- $I \subset \mathbb{R}$ is the *time domain* (a point or an interval) of \mathcal{O} ; and
- $f : \mathbb{R}^2 \times \mathbb{R} \rightarrow \mathbb{R}^2$ is the *transformation function* of \mathcal{O} , which is a time-dependent affinity of the form

$$(x, y; t) \mapsto \begin{pmatrix} a_{11}(t) & a_{12}(t) \\ a_{21}(t) & a_{22}(t) \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} b_1(t) \\ b_2(t) \end{pmatrix},$$

where $a_{ij}(t)$ and $b_i(t)$ are rational functions of t (i.e., of the form $p_1(t)/p_2(t)$, with p_1 and p_2 polynomials in the variable t with rational coefficients) and the determinant of the matrix of the a_{ij} 's differs from zero for all t in I . \square

We remark that this definition guarantees that there is a finite representation of atomic geometric objects by means of the polynomial constraint description of the time-interval, (the cornerpoints of) the reference triangle and the coefficients of the transformation matrices. An atomic geometric object $\mathcal{O} = (T, I, f)$ finitely represents the spatio-temporal object

$$\{(x, y; t) \in \mathbb{R}^2 \times \mathbb{R} \mid t \in I \wedge (\exists x')(\exists y')((x', y') \in T \wedge (x, y) = f(x', y'; t))\},$$

which we denote as $st(\mathcal{O})$. Atomic geometric objects can be combined to more complex geometric objects.

Definition 3 A *geometric object* is a set $\{\mathcal{O}_1, \dots, \mathcal{O}_n\}$ of atomic geometric objects. It represents the spatio-temporal object $\cup_{i=1}^n st(\mathcal{O}_i)$. \square

By definition 3, the atomic geometric objects that compose a geometric object are allowed to overlap in time as well in space. This is a natural definition, but we will see in Section 2.2, that this flexibility in design leads to expensive computations when we want to query spatio-temporal objects represented this way.

We define the *time domain* of a geometric object $\{\mathcal{O}_1, \dots, \mathcal{O}_n\}$ to be the smallest time interval that contains all the time intervals I_i of the atomic geometric objects \mathcal{O}_i (this is the convex closure of these time intervals, denoted by $\overline{\cup_{i=1}^n I_i}$).

Remark that a spatio-temporal object is empty outside the time domain of the geometric object that defines it. Also, within the time domain, a spatio-temporal object is empty at any moment when no atomic object exists.

Example 2 The spatio-temporal object of Example 1 can be represented by the geometric object $\{\mathcal{O}_1, \mathcal{O}_2\}$, where \mathcal{O}_1 is represented by $(T_1, [0, 4], f)$ and \mathcal{O}_2 equals $(T_2, [0, 4], f)$, with T_1 the triangle with corner points $(0, 0)$, $(2, 0)$, $(2, -1)$, T_2 the triangle with corner points $(0, 0)$, $(0, -1)$, $(2, -1)$, and f the transformation $(x, y; t) \mapsto (x + 2t + 2, y - t - 1)$. \square

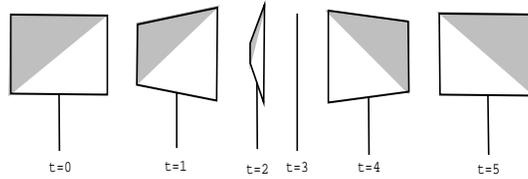


Figure 2: A spatio-temporal object (a traffic sign) shown at six moments.

Example 3 Figure 2 shows a traffic sign at six moments (seen by an observer walking around it). This observation can be described by seven atomic geometric objects. During the interval $[0, 3[$, there exist three atomic objects, two triangles, and one line segment. At the time instant $t = 3$ there exists one atomic object that represents the shape of a line. During the interval $]3, 5]$ there exist three atomic objects, two triangles, and a line segment. \square

To end this subsection, we define the *snapshot* of a spatio-temporal object at a certain moment in time. Snapshots are spatial objects that show what a spatio-temporal object looks like at a certain moment.

Definition 4 Let \mathcal{O} be an atomic object. Let τ_0 be a time moment in the time domain of \mathcal{O} . The *snapshot of \mathcal{O} at time τ_0* , denoted \mathcal{O}^{τ_0} , is the intersection of the spatio-temporal object $st(\mathcal{O})$ with the plane in $(\mathbb{R}^2 \times \mathbb{R})$ defined by $t = \tau_0$, i.e., the plane $\mathbb{R}^2 \times \{\tau_0\}$.

Let $\{\mathcal{O}_1, \dots, \mathcal{O}_n\}$ be a geometric object. The *snapshot of $\{\mathcal{O}_1, \dots, \mathcal{O}_n\}$ at time τ_0* , denoted $\{\mathcal{O}_1, \dots, \mathcal{O}_n\}^{\tau_0}$, is the union of the snapshots at τ_0 of all atomic objects that compose $\{\mathcal{O}_1, \dots, \mathcal{O}_n\}$, i.e., $\cup_{i=1}^n \mathcal{O}_i^{\tau_0}$. \square

As explained in Example 3, Figure 2 shows six snapshots of a geometric object representing a traffic sign seen by a moving observer.

2.2 The Benefits of a Normal Form

As remarked after Definition 3, the atomic objects that compose a geometric object may overlap both in space as in time. As a consequence, it is impossible to answer some very basic queries about a geometric object without a lot of computations.

To know the time domain of a spatio-temporal object, for example, one needs to check all atomic objects that describe it, sort the begin and end points of their time domains, and derive the union of all time domains.

Also, there might be atomic objects that do not contribute at all to the shape of the spatio-temporal object as they are entirely overlapped by other atomic objects. These objects are taken along unnecessarily in computations. Furthermore, two geometric objects that represent the same spatio-temporal

¹For technical reasons, we allow a triangle to degenerate into a line segment or a point.

object can have a totally different representation by means of atomic objects. It is computationally expensive to derive from their different representations that they are actually the same.

These drawbacks can be solved by introducing a *normal form* for geometric objects, that makes their structure more transparent. This normal form should have the property that it is the same for all geometric objects that represent the same spatio-temporal object, independent of their initial representation by means of atomic objects. In this paper, we add the requirement that this normal form should be invariant under affinities. If two geometric objects are the same up to time-dependent affinities, we also want their normal form representation to be the same up to these affinities.

2.3 Affine Triangulation Methods

We end this section with the definition of affine spatial and spatio-temporal triangulation methods.

Definition 5 [Spatial and spatio-temporal triangulation] Let $\{\mathcal{O}_1, \dots, \mathcal{O}_n\}$ be a geometric object and τ_0 be a time moment in the time domain of $\{\mathcal{O}_1, \dots, \mathcal{O}_n\}$.

- A collection of triangles² $\{T_1, T_2, \dots, T_m\}$ in \mathbb{R}^2 is a *triangulation* of the snapshot $\{\mathcal{O}_1, \dots, \mathcal{O}_n\}^{\tau_0}$ if the interiors³ of different T_i are disjoint and the union $\cup_{i=1}^m T_i$ equals $\{\mathcal{O}_1, \dots, \mathcal{O}_n\}^{\tau_0}$.
- A geometric object $\{\mathcal{T}_1, \dots, \mathcal{T}_m\}$ is a *triangulation of a geometric object* $\{\mathcal{O}_1, \dots, \mathcal{O}_n\}$ if for each τ_0 in the time domain of $\{\mathcal{O}_1, \dots, \mathcal{O}_n\}$, $\{\mathcal{T}_1, \dots, \mathcal{T}_m\}^{\tau_0}$ is a triangulation of the snapshot $\{\mathcal{O}_1, \dots, \mathcal{O}_n\}^{\tau_0}$ and if furthermore $st(\{\mathcal{O}_1, \dots, \mathcal{O}_n\}) = st(\{\mathcal{T}_1, \dots, \mathcal{T}_m\})$. \square

We remark that in the second part of Definition 5, at each moment τ_0 in the time domain of $\{\mathcal{O}_1, \dots, \mathcal{O}_n\}$, $\mathcal{T}_i^{\tau_0}$ may be empty (i.e., τ_0 may be outside the time domain of \mathcal{T}_i).

In Figure 3, two stars with their respective triangulations are shown. Note that, although triangulations of spatial sets intuitively are *partitions* of such sets into triangles, they are not partitions in the mathematical sense. Indeed, the elements of the *partition* may have common boundaries. For spatial data, it is customary to allow the elements of a partition to share boundaries (see for example [14]).

A *spatial triangulation method* is a procedure that on input (some representation of) a snapshot of a spatio-temporal object produces (some representation of) a triangulation of this snapshot. A spatio-temporal *triangulation*

²Remark that we consider filled triangles and we allow a triangle to degenerate into a closed line segment or a point.

³We define the *interior* as follows: the interior of a triangle is its topological interior; the interior of a line segment is the segment without endpoints; and the interior of a point is the point itself.

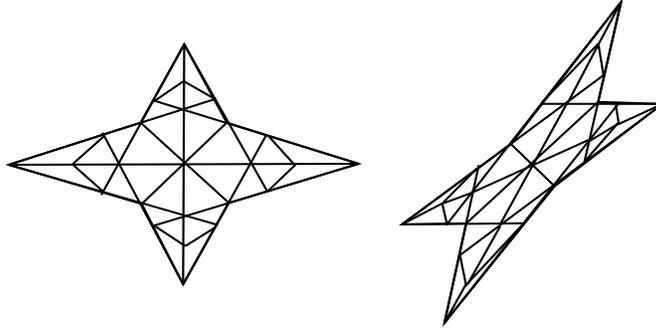


Figure 3: The triangulations of a snapshot (left) and of an affine transformation of the snapshot (right).

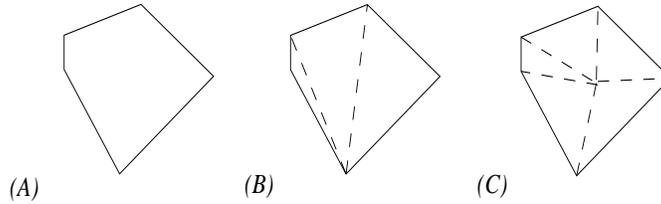


Figure 4: Two different triangulations ((B) and (C)) of a convex polygon ((A)).

method is a procedure that on input (some representation by means of geometric objects of) a spatio-temporal object, produces (some representation by means of geometric objects of) a triangulation of this spatio-temporal object.

Next, we define what it means for such methods to be affine-invariant.

Definition 6 [Affine-invariant triangulation methods] A spatial triangulation method \mathcal{T}_S is called *affine invariant* if and only if for any two snapshots A and B , for which there is an affinity $\alpha : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ such that $\alpha(A) = B$, also $\alpha(\mathcal{T}_S(A)) = \mathcal{T}_S(B)$.

A spatio-temporal triangulation method \mathcal{T}_{ST} is called *affine invariant* if and only if for any geometric objects $\{\mathcal{O}_1, \dots, \mathcal{O}_n\}$ and $\{\mathcal{O}'_1, \dots, \mathcal{O}'_m\}$ for which for each moment τ_0 in their time domains, there is an affinity $\alpha_{\tau_0} : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ such that if $\alpha_{\tau_0}(\{\mathcal{O}_1, \dots, \mathcal{O}_n\}^{\tau_0}) = \{\mathcal{O}'_1, \dots, \mathcal{O}'_m\}^{\tau_0}$, also $\alpha_{\tau_0}(\mathcal{T}_{ST}(\{\mathcal{O}_1, \dots, \mathcal{O}_n\}^{\tau_0})) = \mathcal{T}_{ST}(\{\mathcal{O}'_1, \dots, \mathcal{O}'_m\}^{\tau_0})$. \square

Example 4 Given a convex polygon as shown in (A) of Figure 4. A spatial triangulation method that takes the leftmost of the corner points with the smallest y -coordinates of the polygon and connects it with all other corner points, is not affine invariant. It is not difficult to see that, when an affine transformation is applied to the polygon, another point may become the

leftmost lowest corner point. Part (B) of Figure 4 shows the result of applying this triangulation method to the convex polygon shown in (A).

A triangulation method that computes the barycenter of a convex polygon and connects it with all corner points is affine-invariant. An illustration the output of this method applied to the polygon shown in (A) is shown in (C) of Figure 4. \square

We now propose an affine-invariant spatial triangulation method for spatial figures that are snapshots of geometric objects, or, that can be represented as finite sets of triangles.

3 An Affine-invariant Spatial Triangulation Method

We next propose an affine-invariant triangulation method. Later on, in Section 4, we will use the technique proposed here to construct a spatio-temporal triangulation algorithm. We first explain the intuition behind the triangulation method, and then give the details in Algorithm 1. We illustrate the algorithm with an example, prove its correctness and end with determining the size of the output and the time complexity of the algorithm.

Intuitively, the algorithm is as follows. The input is a snapshot S , given as a finite set of triangles. In Figure 5 (A), for example, a snapshot of a house-like shape is given by four triangles. One of those triangles is degenerated into a line segment (representing the chimney). To make sure that the triangulation is independent of the exact representation of the snapshot by means of triangles, the boundary of the snapshot, i.e., the boundary of the union of the triangles composing S , is computed. For the snapshot of Figure 5, the boundary is shown in (B). The (triangle degenerated into a) line segment contributes to the boundary. Therefore, we label it, the reason for this will become clear in a further stage of the procedure. Also, the (triangles degenerated into) points of the input that are not part of a line segment or real triangle, i.e., the ones contributing to the boundary, are added to the output immediately.

The set of all lines through the edges of the boundary partitions the plane into a set of open convex polygons, open line segments, open (half-) lines and points. The (half-) lines and some of the polygons can be unbounded, so we use the convex hull $\mathcal{CH}(S)$ of the corner points of all triangles in the input as a bounding box. In (C) of Figure 5, the grey area is the area inside of the convex hull. The partition of the area inside the convex hull is computed. The points in this partition are not considered. The points contributing to the boundary were already added to the output in an earlier stage. For each open line segments, it is checked whether it is part of a labelled line segment

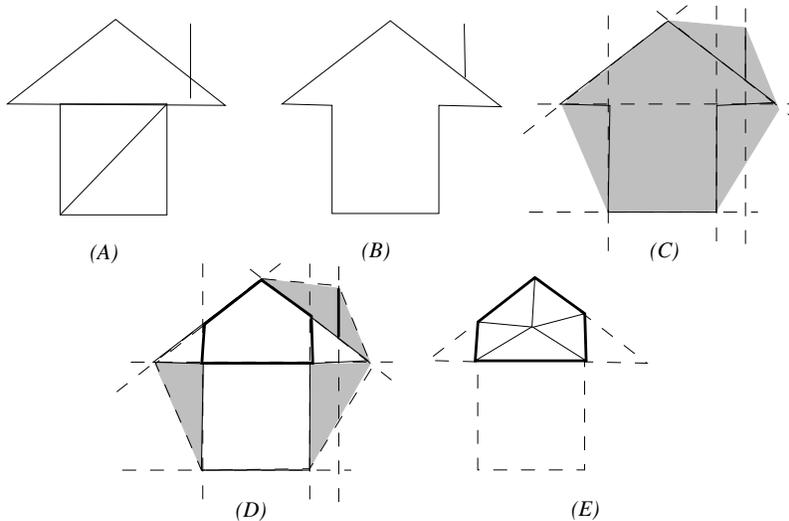


Figure 5: The several steps in the spatial triangulation algorithm.

of the input. Recall that only line segments that contribute to the boundary are labelled in an earlier stage of the algorithm. Only if an open line segment is part of a labelled segment, as is the case for the one printed in bold in Figure 5 (D), its closure (i.e., a closed line segment) is added to the output. For each open polygon in the partition, we compute the polygon that is its closure and triangulate this polygon using its center of mass (see Figure 5 (D) for a polygon in the partition and (E) for its triangulation). Some open polygons are only part of the convex hull of S , but not of the snapshot itself. The polygons shaded in grey in (D) of Figure 5 are an example of such polygons. If a polygon does not belong to S , we do not triangulate it. The triangulations of all other polygons are added to the output. Note that we can decide whether a polygon belongs to the snapshot by first computing the *planar subdivision* $\mathcal{U}(S)$ (which we will define next) of the input snapshot and then test for each open polygon whether its center of mass belongs to the interior of a region or face in the subdivision. We will explain this in more detail when analyzing the complexity of the algorithm.

In the detailed description of the algorithm, we will use some well known techniques. One of those is the *doubly-connected edge list* [10], used to store *planar subdivisions*.

Definition 7 [Planar subdivision] A *planar subdivision* is a subdivision of the plane into labelled regions (or *faces*), edges and vertices, induced by a plane graph. The *complexity of a subdivision* is the sum of the number of vertices, the number of edges and the number of faces it consists of. \square

Next, we describe the *doubly-connected edge list*, a data structure to store planar subdivisions. For this structure, each edge is split into two directed

half-edges. In general, a doubly-connected edge list contains a record for each face, edge and vertex of the planar subdivision.

Definition 8 [Doubly-connected edge list] Given a planar subdivision \mathcal{S} . A *doubly-connected edge list for \mathcal{S}* , denoted $\text{DCEL}(\mathcal{S})$, is a structure containing a record for each face, edge and vertex of the subdivision. These records store the following geometric and topological information:

- (i) The *vertex record* of a vertex \mathbf{a} stores the coordinates of \mathbf{a} and a pointer to an arbitrary half-edge that has \mathbf{a} as its origin;
- (ii) The *face record* of a face f stores a pointer to an arbitrary half-edge on its boundary. Furthermore, for each hole in f , it stores a pointer to an arbitrary half-edge on its boundary;
- (iii) The *half-edge record* of a half-edge e stores five pointers. One to its origin-vertex, one to its twin half-edge, one to the face it bounds, and one to the previous and next half-edge on the boundary on that face.

□

Example 5 Figure 6 shows a planar subdivision in (B) and its topological structure in (C), that is reflected in the doubly-connected edge list represented in Table 2. □

Algorithm 1 (or \mathcal{T}_S) gives the triangulation procedure more formally. The input of this triangulation algorithm is a snapshot S , consisting of a geometric object which we assume to be given as a finite set of (possibly overlapping and possibly degenerated) triangles. We further assume that each triangle is represented as a triple of pairs of coordinates, which are rational numbers.

To shorten and simplify the exposition of Algorithm 1, we assume that S is fully two-dimensional, or equivalently, that points and line segments that are not adjacent to a polygon belonging to S are already in the output. Including their triangulation in the algorithm would make its description tedious, as we would have to add, and consider, more node and edge labels.

We use C programming-style notation for pointers to records and elements of records. For example, Let $\mathbf{a} = (a_1, a_2)$. In the vertex record V_a of \mathbf{a} , $V_a.x = a_1$ and $V_a.y = a_2$. Let e be an edge record. The coordinates of the origin $e \rightarrow \text{origin}$ of e are $e \rightarrow \text{origin} \rightarrow x$ and $e \rightarrow \text{origin} \rightarrow y$.

Before proving the correctness of the algorithm and determining the size of the output and the time complexity of the algorithm, we give an example.

Example 6 Let S be the set $\{T_1, T_2\}$, where T_1 is the triangle with corner points v_1, v_3 and v_4 , and T_2 the triangle with corner points v_2, v_3 and v_4 , as depicted in Figure 6. The doubly-connected edge list corresponding to (C)

Algorithm 1 \mathcal{T}_S (Input: $S = \{T_1, T_2, \dots, T_k\}$, Output: $\{T'_1, T'_2, \dots, T'_\ell\}$)

```

1: Out :=  $\emptyset$ .
2: Compute the set  $\mathcal{B}(S)$  containing all line segments, bounding a triangle
   of the input, that contribute to the boundary of  $S$  (i.e., that contain
   an edge of the boundary). Meanwhile, construct the planar subdivision
    $\mathcal{U}(S)$  induced by the triangles composing  $S$ .
3: Compute the convex hull  $\mathcal{CH}(S)$  of  $S$ .
4: Construct the doubly connected edge list  $\text{DCEL}(S)$ , induced by the pla-
   nar subdivision defined by the lines through the segments of  $\mathcal{B}(S)$ , using
    $\mathcal{CH}(S)$  as a bounding box.
5: while there are any unvisited half-edges in  $\text{DCEL}(S)$  left do
6:   Let  $e$  be an unvisited edge.
7:    $\Sigma_x := 0, \Sigma_y := 0, \text{count} := 0, \text{Elist} := \emptyset$ .
8:   while  $e$  is unvisited do
9:     Mark  $e$  with the label visited.
10:     $\text{Elist} := \text{Elist} \cup \{(e \rightarrow \text{origin}, e \rightarrow \text{next} \rightarrow \text{origin})\}, \Sigma_x := \Sigma_x + e \rightarrow$ 
        $\text{origin} \rightarrow x, \Sigma_y := \Sigma_y + e \rightarrow \text{origin} \rightarrow y, \text{count} := \text{count} + 1$ .
11:     $e := e \rightarrow \text{next}$ .
12:   end while
13:    $\mathbf{x} := (\frac{\Sigma_x}{\text{count}}, \frac{\Sigma_y}{\text{count}})$ .
14:   if the point  $\mathbf{a}$  in  $\mathbf{x}$  belongs to a face of  $\mathcal{U}(S)$  then
15:     for all elements  $(\mathbf{a}_s, \mathbf{a}_e)$  of  $\text{Elist}$  do
16:        $\text{Out} := \text{Out} \cup \{T_{\mathbf{a}\mathbf{a}_s\mathbf{a}_e}\}$ , where  $T_{\mathbf{a}\mathbf{a}_s\mathbf{a}_e}$  is the (closed) triangle with
         corner points  $\mathbf{a}, \mathbf{a}_s$  and  $\mathbf{a}_e$ .
17:     end for
18:   end if
19: end while
20: return Out.

```

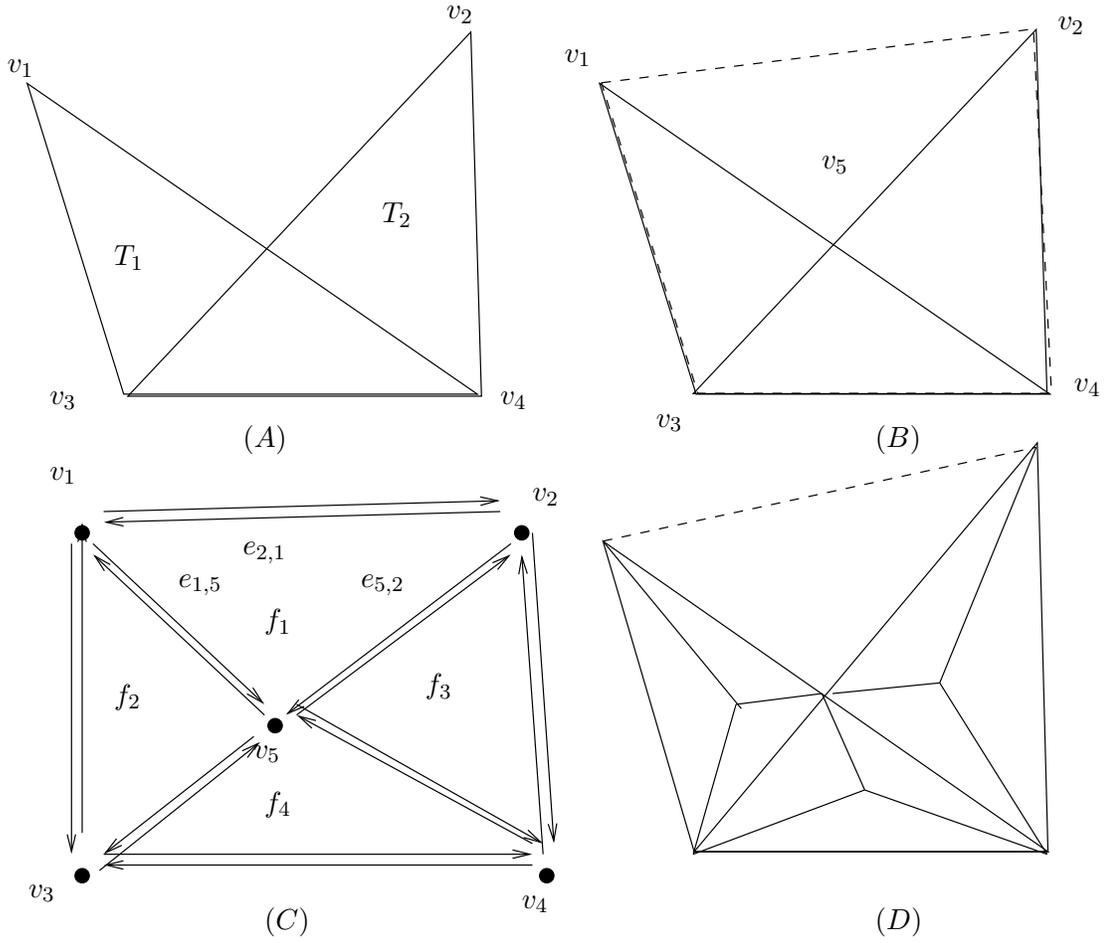


Figure 6: The different steps of Algorithm 1 applied to $S = \{T_1, T_2\}$. In this example, all boundary segments of all triangles of S contribute to the boundary of S . The line arrangement induced by the carriers of the edges of the input triangles is bounded by the convex hull of the input in (B). A doubly-connected edge list is constructed out of the line arrangement, storing its topological structure in (C). Finally, the triangulation is computed in (D).

Half-edge	Origin	Twin	IncidentFace	Next	Prev
$e_{1,2}$	v_1	$e_{2,1}$	f_5	$e_{2,4}$	$e_{3,1}$
$e_{2,1}$	v_2	$e_{1,2}$	f_1	$e_{1,5}$	$e_{5,2}$
$e_{1,3}$	v_1	$e_{3,1}$	f_2	$e_{3,5}$	$e_{5,2}$
$e_{3,1}$	v_3	$e_{1,3}$	f_5	$e_{1,2}$	$e_{4,3}$
$e_{1,5}$	v_1	$e_{5,1}$	f_1	$e_{5,2}$	$e_{2,1}$
$e_{5,1}$	v_5	$e_{1,5}$	f_2	$e_{1,3}$	$e_{3,5}$
$e_{2,4}$	v_2	$e_{4,2}$	f_5	$e_{4,3}$	$e_{1,2}$
$e_{4,2}$	v_4	$e_{2,4}$	f_3	$e_{2,5}$	$e_{5,4}$
$e_{2,5}$	v_2	$e_{5,2}$	f_3	$e_{5,4}$	$e_{4,2}$
$e_{5,2}$	v_5	$e_{2,5}$	f_1	$e_{2,1}$	$e_{1,5}$
$e_{3,4}$	v_3	$e_{4,3}$	f_4	$e_{4,5}$	$e_{5,3}$
$e_{4,3}$	v_4	$e_{3,4}$	f_5	$e_{3,1}$	$e_{2,4}$
$e_{3,5}$	v_3	$e_{5,3}$	f_2	$e_{5,1}$	$e_{1,3}$
$e_{5,3}$	v_5	$e_{3,5}$	f_4	$e_{3,4}$	$e_{4,5}$
$e_{4,5}$	v_4	$e_{5,4}$	f_4	$e_{5,3}$	$e_{3,4}$
$e_{5,4}$	v_5	$e_{4,5}$	f_3	$e_{4,2}$	$e_{2,5}$

Table 2: The half-edge records of the doubly-connected edge list corresponding to Figure 6.

is shown in Table 2. We omitted the structures for vertices and faces, as we don't need them for the second part of the algorithm.

After the doubly-connected edge list is constructed, we create and output the triangles. This is done by visiting all half-edges once. Suppose we start with $e_{2,1}$. The next-pointers lead to $e_{1,5}$ and $e_{5,2}$. The next pointer of the last one points to $e_{2,1}$, which we visited already. This means we visited all edges of one polygon. The center of mass can now be computed and the triangles added to the output. This is done for all polygons that are part of the input. In this example, the polygon with corner points v_1 , v_5 and v_2 will not be triangulated, as it is not part of the input. The algorithm stops when there are no more unvisited edges left. \square

Note that, as an optimization, we could decide to not triangulate faces that are triangles already. This does not influence the complexity results, however. Therefor, and also for a shorter and more clear exposition, we formulated the algorithm in a more general form.

We now prove compute the complexity of both the output and execution time of the triangulation method described in Algorithm 1 and afterwards show that it is affine-invariant. First, we show that \mathcal{T}_S is indeed a triangulation method.

Property 1 (Algorithm 1 is a triangulation method) Let S be a snapshot. The output $\mathcal{T}_S(S)$ of Algorithm 1 applied to S is a triangulation of S .

Proof. Let the set of triangles $\{T_1, T_2, \dots, T_k\}$ determine a snapshot S . It is easy to see that the output $\mathcal{T}_S(S) = \{T'_1, T'_2, \dots, T'_\ell\}$ of \mathcal{T}_S is a triangulation. By construction, $\mathcal{T}_S(S)$ is a set of triangles that either have no intersection, or share a corner point or bounding segment. It is clear from the algorithm that $\bigcup_{i=1}^k T_i = \bigcup_{j=1}^\ell T'_j$, because each triangle in $\mathcal{T}_S(S)$ is tested for membership of S . We are also sure that S is covered by the output, because initially, the convex hull of S is triangulated, which contains S . \square

Property 2 (Quadratic output complexity) Let a snapshot S be given by the set $\{T_1, T_2, \dots, T_m\}$, consisting of m triangles. The triangulation $\mathcal{T}_S(S)$, where \mathcal{T}_S is the triangulation method described in Algorithm 1, contains $O(m^2)$ triangles.

Proof. It is well-known (see, e.g., [8]), that an arrangement of m lines in the plane results in a subdivision of the plane containing $O(m^2)$ lines, $O(m^2)$ edges and $O(m^2)$ faces. It follows that the structure $\text{DCEL}(\mathcal{S})$ will contain $O(m^2)$ half-edges, i.e., two half-edges for each edge in the arrangement. In the worst case scenario, when all faces of the partition of the bounding box belong to S , one triangle is added to the output for each half-edge in $\text{DCEL}(\mathcal{S})$ (connecting that half-edge with the center of mass of the face it bounds). Therefore, the output contains $O(m^2)$ triangles. \square

In the following analysis of the running time of Algorithm 1, we assume that triangles are represented as triples of points, and that a point is represented as a pair of rational or algebraic numbers. We further assume that all basic arithmetic operations on coordinates of points require constant time.

Property 3 ($O(m^2 \log m)$ running time) Let a snapshot S be given by the set $\{T_1, T_2, \dots, T_m\}$, consisting of m triangles. The triangulation method \mathcal{T}_S , described in Algorithm 1, computes the triangulation $\mathcal{T}_S(S)$ of S in time $O(m^2 \log m)$.

Proof. Let a snapshot S be given by the set $\{T_1, T_2, \dots, T_m\}$. Using a plane-sweep algorithm [11], we compute both the list of segments contributing to the boundary of S and the planar subdivision $\mathcal{U}(S)$ induced by $\bigcup_{i=1}^m T_i$. This takes $O(m^2 \log m)$, as there are at most m^2 intersection points between boundary segments of triangles of S .

The m triangles composing S together have at most $3m$ different corner points. Computing the convex hull of m points in the plane can be done in time $O(m \log m)$ (see [9]). The same authors propose, in [8], an algorithm to compute a doubly-connected edge list, representing an arrangements of m lines, in time $O(m^2)$. We next show that the changes we make to this algorithm do not influence its running time. So, as $\mathcal{B}(S)$ contains at most all $3m$ line segments, it induces an arrangement of at most $3m$ lines. Hence, Step 3 of Algorithm 1 also takes time $O(m^2)$.

We changed the original algorithm [8] for computing the doubly-connected edge list of an arrangement of lines as follows:

- (i) We computed the *convex hull* of the input to serve as a bounding box instead of an axis-parallel rectangle containing all intersection points of the arrangement. The complexity of computing such an axis-parallel rectangle is higher ($O(m^2)$) than that of computing the convex hull ($O(m \log m)$).
- (ii) The cost of constructing the doubly-connected edge list of the convex hull is $O(m)$, as the convex hull contains at most $3m$ corner points and the algorithm for computing it, as described in [9], already outputs the corner points of the convex hull in circular order. In the original algorithm [8] with an axis-parallel bounding rectangle, computing the doubly-connected edge list of this rectangle only takes constant time. This extra time does, however, not affect the overall complexity.
- (iii) The next step of both algorithms involves finding the intersection points between the lines to be inserted and the partial arrangement induced by the previously inserted lines. In the original algorithm, this is easier only for the intersection of a line with the bounding box. For the intersections with all other lines in the arrangement, the cost is the same.

The next part of Algorithm 1 (starting from Line 5) takes time $O(m^2 \log m)$. Each half-edge of the doubly-connected edge list is visited only once. Also, each half-edge is only inserted once into the set *Elist*, and consulted only once therein to create a triangle. As an arrangement of m lines in the plane results in $O(m^2)$ edges, the number of half-edges in $\text{DCEL}(\mathcal{S})$ also is $O(m^2)$. We can, in time $O(m^2)$, preprocess $\mathcal{U}(S)$ into a structure that allows point location in $O(\log m)$ time [13]. Therefor, testing for each of the $O(m^2)$ centers of mass whether they are part of the input takes $O(m^2 \log m)$. We can conclude that all parts of Algorithm 1 run in time $O(m^2 \log m)$. \square

Table 3 summarizes the computational complexity of the various parts of Algorithm 1.

Property 4 (\mathcal{T}_S is affine-invariant) The triangulation method \mathcal{T}_S is affine-invariant.

Proof. According to the definition of affine-invariance of spatial triangulation methods (Definition 6), we have to prove the following. Let A be a snapshot given by the set of triangles $\{T_{a,1}, T_{a,2}, \dots, T_{a,k}\}$ and B be a snapshot given by the set of triangles $\{T_{b,1}, T_{b,2}, \dots, T_{b,\ell}\}$, such that there exists an affinity $\alpha : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ for which $B = \alpha(A)$. Then, for each triangle T of $\mathcal{T}_S(A)$, it holds that the triangle $\alpha(T)$ is a triangle of $\mathcal{T}_S(B)$.

Line(s)	Step	Time complexity
2	Compute $\mathcal{B}(S)$ and $\mathcal{U}(S)$	$O(m^2 \log m)$
3	Compute $\mathcal{CH}(S)$	$O(m \log m)$
4	Compute $\text{DCEL}(\mathcal{S})$	$O(m^2)$
5 – 19	Polygon extraction and triangulation	$O(m^2 \log m)$
	Overall time complexity	$O(m^2 \log m)$

Table 3: The time complexity of the various parts of Algorithm 1, when the input is a snapshot represented by n triangles.

We prove this by going through the steps of the triangulation procedure \mathcal{T}_S . Let A and B be as above.

The convex hull and boundary of spatial figures are both affine-invariant (more specific, the boundary is a topological invariant). Intersection points between lines and the order of intersection points on one line with other lines are affine-invariant (even topological invariant). The subdivision of the convex hull $\mathcal{CH}(B)$ of B induced by the arrangement of lines through the boundary of B is hence the image under α of the subdivision of the convex hull $\mathcal{CH}(A)$ of A induced by the arrangement of lines through the boundary of A . The doubly-connected edge list only stores topological information about the arrangement of lines, i.e., which edges are incident to which vertices and faces. Naturally, this information is preserved by affine transformations. The center of mass of a convex polygon is an affine invariant. Finally, the fact that a triangle is inside the boundary of the input and the fact that it is not are both affine-invariant. This completes the proof. \square

Summarizing this section, we proposed a spatial triangulation method that, given a snapshot consisting of m triangles, returns an affine-invariant triangulation of this snapshot containing $O(m^2)$ triangles, in time $O(m^2 \log m)$.

We remark here that the idea of using carriers of boundary segments to partition figures was also used in an algorithm to decompose semi-linear sets by Dumortier, Gyssens, Vandeurzen and Van Gucht [12]. Their algorithm is not affine-invariant, however.

In the next section, we will use the affine triangulation for snapshots to construct a triangulation of geometric objects.

4 An Affine-invariant Spatio-temporal Triangulation Method

In this section, we present an spatio-temporal triangulation algorithm that takes as input a geometric object, i.e., a finite set of atomic geometric objects. We will adapt the spatial triangulation method \mathcal{T}_S , described in Algorithm 1, for time-dependent data.

The proposed spatio-temporal triangulation algorithm \mathcal{T}_{ST} will have three main construction steps. First, in the *partitioning step*, the time domain of the geometric object will be partitioned into a set of points and open time intervals. For each element of this partition, all its snapshots have an *isomorphic triangulation*, when computed by the method \mathcal{T}_S . We refer to Definition 9 below for a formal definition of this isomorphism. Second, in the *triangulation step*, the spatio-temporal triangulation is computed for each element in the time partition, using the fact that all snapshots have *isomorphic triangulations*. Third, in the *merge step*, we merge objects when possible, to obtain a unique (and minimal) triangulation.

We will start this section by defining isomorphic triangulations. Then we explain the different steps of the algorithm for computing a spatio-temporal affine-invariant triangulation of geometric objects separately. We illustrate the algorithm with an example and end with some properties of the triangulation.

Intuitively, two snapshots S_1 and S_2 are called \mathcal{T}_S -isomorphic if the triangles in $\mathcal{T}_S(S_1) \cup \mathcal{T}_S(\mathcal{CH}(S_1) \setminus S_1)$ and $\mathcal{T}_S(S_2) \cup \mathcal{T}_S(\mathcal{CH}(S_2) \setminus S_2)$ have the same (topological) adjacency graph. In particular, if S_1 and S_2 are equal up to an affinity of \mathbb{R}^2 , then they are \mathcal{T}_S -isomorphic.

Definition 9 [\mathcal{T}_S -isomorphic snapshots] Let S_1 and S_2 be two snapshots of a geometric object. We say that S_1 and S_2 are \mathcal{T}_S -isomorphic, denoted $S_1 \equiv_{\mathcal{T}_S} S_2$, if there exists a bijective mapping $h : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ with the following property: A triangle $T = (\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3)$ of $\mathcal{T}_S(S_1)$ is incident to the triangles $T_{1,2}$, $T_{2,3}$ and $T_{3,1}$ (where each $T_{i,((i+1) \bmod 3)}$ is either a triangle of $\mathcal{T}_S(S_1)$ that shares the segment $\mathbf{a}_i \mathbf{a}_{((i+1) \bmod 3)}$ with T , a triangle of $\mathcal{T}_S(\mathcal{CH}(S_1) \setminus S_1)$ that shares the segment $\mathbf{a}_i \mathbf{a}_{((i+1) \bmod 3)}$ with T , or is ϵ , which means that no triangle shares that boundary segment with T) if and only if, the triangle $h(T) = (h(\mathbf{a}_1), h(\mathbf{a}_2), h(\mathbf{a}_3))$ belongs to $\mathcal{T}_S(S_2)$ and is bounded by $h(T_{1,2})$, $h(T_{2,3})$ and $h(T_{3,1})$. Moreover, if $T_{i,((i+1) \bmod 3)}$ is a triangle of $\mathcal{T}_S(S_1)$, then $h(T_{i,((i+1) \bmod 3)})$ is a triangle of $\mathcal{T}_S(S_2)$ that shares the line segment $h(\mathbf{a}_i)h(\mathbf{a}_{((i+1) \bmod 3)})$ with $h(T)$, if $T_{i,((i+1) \bmod 3)}$ is a triangle of $\mathcal{T}_S(\mathcal{CH}(S_1) \setminus S_1)$, then $h(T_{i,((i+1) \bmod 3)})$ is a triangle of $\mathcal{T}_S(\mathcal{CH}(S_2) \setminus S_2)$ that shares the line segment $h(\mathbf{a}_i)h(\mathbf{a}_{((i+1) \bmod 3)})$ with $h(T)$ and if $T_{i,((i+1) \bmod 3)}$ equals ϵ , then so does $h(T_{i,((i+1) \bmod 3)})$. \square

Example 7 The triangulations shown in Figure 3 are \mathcal{T}_S -isomorphic to each other. In Figure 7, all snapshots shown except the one at time moment $t = 3$ are \mathcal{T}_S -isomorphic. The snapshot at time moment $t = 3$ is clearly not isomorphic to the others, since it consists only of one line segment. \square

Remark that for Figure 3, the mapping h is an affinity. In Figure 7, this is not the case.

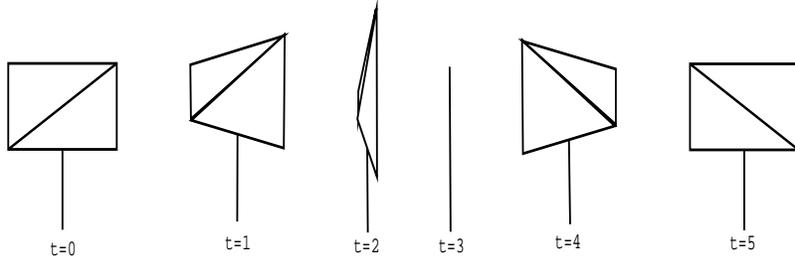


Figure 7: Snapshots of a traffic sign as seen by an observer circularly moving around it.

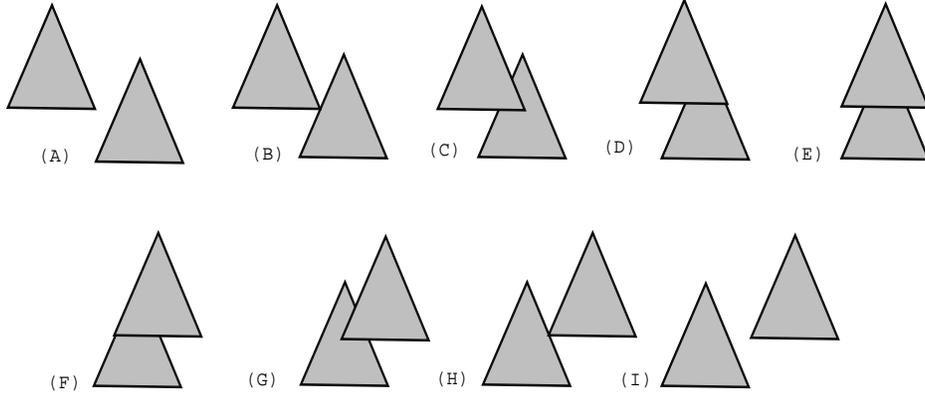


Figure 8: The snapshots at time moments $t = \frac{1}{4}$ (A), $t = \frac{1}{2}$ (B), $t = 1$ (C), $t = \frac{3}{2}$ (D), $t = 2$ (E), $t = \frac{5}{2}$ (F), $t = 3$ (G), $t = \frac{7}{2}$ (H) and $t = 4$ (I) of the geometric object of Example 8.

Now, we introduce a spatio-temporal triangulation method \mathcal{T}_{ST} that constructs a time-dependent affine triangulation of spatio-temporal objects that are represented by geometric objects. We will explain its three main steps, i.e., the partitioning step, the triangulation step and the merge step separately in the next subsections.

We will illustrate each step on the following example.

Example 8 Let $\mathcal{O} = \{\mathcal{O}_1, \mathcal{O}_2\}$ be a geometric object, where \mathcal{O}_1 is given as $(((-1, 0), (1, 0), (0, 2)), [0, 4], Id)$ and \mathcal{O}_2 is given as $(((-3, 1), (-1, 1), (-2, 3)), [0, 4], f)$ and f is the mapping given by $(x, y, t) \mapsto (x + t, y)$. Figure 8 shows the snapshots of \mathcal{O} at time moments $t = \frac{1}{4}$ (A), $t = \frac{1}{2}$ (B), $t = 1$ (C), $t = \frac{3}{2}$ (D), $t = 2$ (E), $t = \frac{5}{2}$ (F), $t = 3$ (G), $t = \frac{7}{2}$ (H) and $t = 4$ (I). \square

Let $\mathcal{O} = \{\mathcal{O}_1 = (S_1, I_1, f_1), \mathcal{O}_2 = (S_2, I_2, f_2), \dots, \mathcal{O}_m = (S_m, I_m, f_m)\}$ be a geometric object. We assume that the S_i are given as triples of points (i.e., pairs of rational or algebraic numbers), the I_i as structures containing two rational or algebraic numbers and two flags (indicating whether the interval is closed on the left or right side) and, finally, the f_i are affinities given by

rational functions, i.e., fractions of polynomials with integer coefficients (that we assume to be given in dense or sparse representation), for $i = 1, \dots, m$.

4.1 The Partitioning Step.

Let $\mathcal{O} = \{\mathcal{O}_1 = (S_1, I_1, f_1), \mathcal{O}_2 = (S_2, I_2, f_2), \dots, \mathcal{O}_m = (S_m, I_m, f_m)\}$ be a geometric object. In the first step of \mathcal{T}_{ST} , the time domain I of \mathcal{O} , i.e., the convex closure $\overline{\bigcup_{i=1}^m I_i}$ of the union of all the time domains $I_i (i = 1, \dots, m)$ is partitioned in such a way that, for each element of that partition, all its snapshots are \mathcal{T}_S -isomorphic.

Below, we refer to the set of lines that intersect the border of $f_i(S_i, \tau)$ in infinitely many points, the set of *carriers of the snapshot* $f_i(S_i, \tau)$ and denote it $car(f_i(S_i, \tau))$, ($i = 1, \dots, m$).

In [5], we defined the *finite time partition* \mathcal{P} of the time domain of two atomic objects in such a way that for each element P of \mathcal{P} , the carrier sets of each snapshot of P are topologically equivalent. This definition can easily be extended to an arbitrary number of atomic objects. Also the property stating that the finite time partition exists, still holds in the extended setting.

Definition 10 [Generalized finite time partition] We call a *finite time partition* of a geometric object $\mathcal{O} = \{\mathcal{O}_1, \mathcal{O}_2, \dots, \mathcal{O}_m\}$ any partition of the interval $I = \overline{\bigcup_{i=1}^m I_i}$ into a finite number of time intervals J_1, \dots, J_k such that for any $\tau, \tau' \in J_\ell$ (and all $1 \leq \ell \leq k$), $\bigcup_{i=1}^m car(f_i(S_i, \tau))$ and $\bigcup_{i=1}^m car(f_i(S_i, \tau'))$ are topologically equivalent sets in \mathbb{R}^2 . \square

Here, two subsets A and B of \mathbb{R}^2 are called *topologically equivalent* when there exists an orientation-preserving homeomorphism h of \mathbb{R}^2 such that $h(A) = B$.

The proof of the following property follows the lines of a proof in [5].

Property 5 (Existence of a generalized finite time partition) *Let $\mathcal{O} = \{\mathcal{O}_1, \mathcal{O}_2, \dots, \mathcal{O}_m\}$ be a geometric object. There exists a finite time partition of \mathcal{O} .* \square

We now proceed with the partitioning step of the spatio-temporal triangulation algorithm. In this step, a generalized finite time partition of \mathcal{O} is computed, using the information of the *time-dependent* carriers of the atomic objects in \mathcal{O} . Each time an intersection point between two or more time-dependent carriers starts or ceases to exist, or when intersection points change order along a line, a new time interval of the partition is started. Given three *continuously moving* lines, the intersection points of the first line with the two other lines only change order along the first line, if there

Algorithm 2 Partition (Input: $\mathcal{O} = \{\mathcal{O}_1, \dots, \mathcal{O}_n\}$, Output: $\chi = (\tau_1, \tau_2, \dots, \tau_m)$)

- 1: Let $\chi = (\tau_1 \leq \tau_2 \leq \dots \leq \tau_k)$ (with $2 \leq k \leq 2n$) be a sorted list of time moments that appear either as a begin or endpoint of I_i for any of the objects $\mathcal{O}_i = (S_i, I_i, f_i)$, $1 \leq i \leq n$.
 - 2: $\mathcal{C} = \emptyset$.
 - 3: **for all** atomic objects $\mathcal{O}_i = (S_i, I_i, f_i)$, $1 \leq i \leq n$ **do**
 - 4: Add the new atomic objects $(S_{i,1}, I_i, f_i)$, $(S_{i,2}, I_i, f_i)$ and $(S_{i,3}, I_i, f_i)$ to \mathcal{C} , where $S_{i,1}$, $S_{i,2}$ and $S_{i,3}$ are the boundary segments of S_i .
 - 5: **end for**
 - 6: **for all** pairs of objects (S_{i,ℓ_1}, I_i, f_i) and (S_{j,ℓ_2}, I_j, f_j) of \mathcal{C} ($1 \leq i < j \leq n$; $1 \leq \ell_1, \ell_2 \leq 3$) **do**
 - 7: **if** $I_i \cap I_j \neq \emptyset$ **then**
 - 8: Compute the end points of the intervals during which the intersection of the carriers of both time-dependent line segments does exist. Add those such end points that lie within the interval $I_i \cap I_j$ to χ , in a sorted way.
 - 9: **end if**
 - 10: **end for**
 - 11: **for all** triples of objects (S_{i,ℓ_1}, I_i, f_i) , (S_{j,ℓ_2}, I_j, f_j) and (S_{k,ℓ_3}, I_k, f_k) of \mathcal{C} ($1 \leq i < j < k \leq n$; $1 \leq \ell_1, \ell_2, \ell_3 \leq 3$) **do**
 - 12: **if** $I_i \cap I_j \cap I_k \neq \emptyset$ **then**
 - 13: Compute the end points of the intervals during which the carriers of the three time-dependent line segments intersect in one point. Add those such end points that lie within the interval $I_i \cap I_j \cap I_k$ to χ , in a sorted way.
 - 14: **end if**
 - 15: **end for**
 - 16: Return χ .
-

Element	Carrier
$\mathcal{O}_{c,1} = (((-1, 0), (1, 0)), [0, 4], Id)$	$y = 0$
$\mathcal{O}_{c,2} = (((-1, 0), (0, 2)), [0, 4], Id)$	$y = 2x + 2$
$\mathcal{O}_{c,3} = (((0, 2), (1, 0)), [0, 4], Id)$	$y = -2x + 2$
$\mathcal{O}_{c,4} = (((-3, 1), (-1, 1)), [0, 4], f)$	$y = 1$
$\mathcal{O}_{c,5} = (((-3, 1), (-2, 3)), [0, 4], f)$	$y = 2x + 7 - 2t$
$\mathcal{O}_{c,6} = (((-2, 3), (-1, 1)), [0, 4], f)$	$y = -2x - 1 + 2t$

Table 4: The elements of the list \mathcal{C} during the execution of the partitioning algorithm (Algorithm 2) on the geometric object from Example 9.

exists a moment where all three lines intersect in one point. Algorithm 2 describes the partitioning step in detail.

We will show later that the result of the generalized finite time partition is a set of intervals during which all snapshots are \mathcal{T}_S -isomorphic. This partition is, however, not the coarsest possible partition having this property, because there might be atomic objects that, during some time, are completely overlapped by other atomic objects. Therefore, we will later, after the triangulation step, again merge elements of the generalized finite time partition, whenever possible.

We illustrate Algorithm 2 on the geometric object of Example 8.

Example 9 Recall from Example 8 that $\mathcal{O} = \{\mathcal{O}_1, \mathcal{O}_2\}$, where \mathcal{O}_1 is given as $(((-1, 0), (1, 0), (0, 2)), [0, 4], Id)$ and \mathcal{O}_2 is given as $(((-3, 1), (-1, 1), (-2, 3)), [0, 4], f)$ and f is the affinity mapping triples (x, y, t) to pairs $(x + t, y)$.

We now illustrate the partitioning algorithm on input \mathcal{O} . First, the list χ will contain the time moments 0 and 4. The list \mathcal{C} will contain six elements. Table 4 shows these segments and the formulas describing their time-dependent carriers. All pairs of segments have an intersection that exists always, except for the pairs $(\mathcal{O}_{c,2}, \mathcal{O}_{c,5})$, $(\mathcal{O}_{c,3}, \mathcal{O}_{c,6})$ and $(\mathcal{O}_{c,1}, \mathcal{O}_{c,4})$. The intersections of $\mathcal{O}_{c,2}$ with $\mathcal{O}_{c,5}$ and $\mathcal{O}_{c,3}$ with $\mathcal{O}_{c,6}$ exist only at respectively $t = \frac{5}{2}$, $t = \frac{3}{2}$. The segments $\mathcal{O}_{c,1}$ and $\mathcal{O}_{c,4}$ never intersect. Of all possible triples of carriers, only two triples have a common intersection within the interval $[0, 4]$. The carriers of $\mathcal{O}_{c,2}$, $\mathcal{O}_{c,4}$ and $\mathcal{O}_{c,6}$ intersect at $t = \frac{1}{2}$ and the carriers of $\mathcal{O}_{c,3}$, $\mathcal{O}_{c,4}$ and $\mathcal{O}_{c,5}$ intersect at $t = \frac{7}{2}$. The partitioning step will hence return the list

$$\chi = (0, \frac{1}{2}, \frac{3}{2}, \frac{5}{2}, \frac{7}{2}, 4).$$

□

We analyze both the output complexity and sequential time complexity of the partition step. First remark that the product of ℓ univariate polynomials of degree d is a polynomial of degree ℓd . Let the transformation function of an

atomic object consists of rational coefficients, being fractions of polynomials of degree at most d . It follows that the time-dependent line segments and carriers can be defined using fractions of polynomials in t of degree $O(d)$. Also, the time-dependent intersection point of two such carriers and the time-dependent cross-ratio of an intersection point compared to two moving end points of a segment, can be defined using fractions of polynomials in t of degree $O(d)$.

Property 6 (Partition: output complexity) Given a geometric object $\mathcal{O} = \{\mathcal{O}_1 = (S_1, I_1, f_1), \mathcal{O}_2 = (S_2, I_2, f_2), \dots, \mathcal{O}_n = (S_n, I_n, f_n)\}$ consisting of n atomic objects. Let d be the maximal degree of any polynomial in the definition of the transformation functions $f_i, 1 \leq i \leq n$. The procedure **Partition**, as described in Algorithm 2, returns a partition of $I = \bigcup_{i=1}^n I_i$ containing $O(n^3d)$ elements.

Proof. It is clear that the list χ contains $O(n)$ elements after Line 1 of Algorithm 2. Indeed, at most two elements are added for each atomic object. The list \mathcal{C} will contain at most $3n$ elements. For each atomic object with a reference object that is a “real” triangle, 3 elements will be added to \mathcal{C} . In the case that one or more corner points coincide, one or two objects will be added to \mathcal{C} .

Now we investigate the number of time moments that will be inserted to χ while executing the for-loop starting at Line 6 of Algorithm 2. The intervals during which the intersection of two time-dependent carriers exists are computed. The intersection of two time-dependent line segments doesn't exist at time moments where the denominator of the rational function defining it is zero. Because this denominator always is a polynomial P in t , it has at most $\deg(P)$ zeroes, where $\deg(P)$ denotes the degree of P . Accordingly, at most $\deg(P) = O(d)$ elements will be added to χ . Hence, in total, $O(n^2d)$ time moments are added in this step.

For the intersections of three carriers, a similar reasoning can be used. Hence, during the execution of the for-loop starting at Line 11 of Algorithm 2, $O(n^3d)$ elements are added to χ .

We can conclude that the list χ will contain $O(n^3d)$ elements. \square

Now we analyze the time complexity of **Partition**. We first point out that finding all roots of an univariate polynomial of degree d , with accuracy ϵ can be done in time $O(d^2 \log d \log \log(\frac{1}{\epsilon}))$ [26]. We will use the abbreviation $z(d, \epsilon)$ for the expression $O(d^2 \log d \log \log(\frac{1}{\epsilon}))$. Note also that, although the product of two polynomials of degree d is a polynomial of degree $2d$, the computation of the product takes time $O(d^2)$. To keep the proofs of the complexity results as readable as possible, we will consider the complexity of any manipulation on polynomials (computing zeros, adding or multiplying) to be $z(d, \epsilon)$, where a precision of ϵ is obtained.

Property 7 (Partition: computational complexity) Given a geometric object $\mathcal{O} = \{\mathcal{O}_1 = (S_1, I_1, f_1), \mathcal{O}_2 = (S_2, I_2, f_2), \dots, \mathcal{O}_n = (S_n, I_n, f_n)\}$ consisting of n atomic objects. Let d be the maximal degree of any polynomial in the definition of the transformation functions $f_i, 1 \leq i \leq n$ and let ϵ be the desired precision for computing the zeros of polynomials. The procedure **Partition**, as described in Algorithm 2, returns a partition of $I = \overline{\bigcup_{i=1}^n I_i}$ in time $O(n^3(z(d, \epsilon) + d \log n))$.

Proof. Let $\mathcal{O} = \{\mathcal{O}_1 = (S_1, I_1, f_1), \mathcal{O}_2 = (S_2, I_2, f_2), \dots, \mathcal{O}_n = (S_n, I_n, f_n)\}$ be a geometric object. Let d be the maximum degree of any of the polynomials used in the definition of the functions $f_i, 1 \leq i \leq n$.

Constructing the initial list χ , on Line 1, takes time $O(n \log n)$ (it is well known that the inherent complexity of sorting a list of n elements is $O(n \log n)$). Computing the set \mathcal{C} can be done in time $O(nd)$: all n elements of \mathcal{O} are considered, and the time needed to copy the transformation functions f_i depends on the maximal degree the polynomials defining them have. Recall that \mathcal{C} contains at most $3n$ elements.

The first for-loop, starting at Line 6 of Algorithm 2 is executed $O(n^2)$ times. One execution of its body takes $z(d, \epsilon)$. Indeed, computing the formula representing the time-dependent intersection, checking whether its denominator is always zero and finding the zeros of the denominator (a polynomial of degree linear in d) have all time complexity $z(d, \epsilon)$. Therefore, the first for-loop takes time $O(n^2 z(d, \epsilon))$ in total.

The second for-loop has time complexity $O(n^3 z(d, \epsilon))$. The reasoning here is the same as for the previous for-loop.

Finally, sorting the list χ , which contains $O(n^3 d)$ elements at the end, requires $O(n^3 d \log(nd))$.

If we summarize the complexity of all the separate steps, we obtain $O(n^3(z(d, \epsilon) + d \log n))$. \square

We now proceed with the triangulation step.

4.2 The Triangulation Step.

Starting with a geometric object $\mathcal{O} = \{\mathcal{O}_1 = (S_1, I_1, f_1), \mathcal{O}_2 = (S_2, I_2, f_2), \dots, \mathcal{O}_n = (S_n, I_n, f_n)\}$, the partitioning algorithm identifies a list χ of time moments that is used to partition the time domain $I = \overline{\bigcup_{i=1}^n I_i}$ of \mathcal{O} into points and open intervals. For each element in that partition (point or open interval), we now triangulate the part of \mathcal{O} restricted to that point or open interval.

The triangulation of the snapshots of \mathcal{O} at the time moments in χ is straightforward. For each of the time moments τ of χ , the spatial triangulation method \mathcal{T}_S is applied to the snapshot \mathcal{O}^τ . For each of the triangles T in $\mathcal{T}_S(\mathcal{O}^\tau)$, an atomic object is constructed with T as reference object, the singleton $\{\tau\}$ as time domain and the identity as its transformation function.

The triangulation of the parts of \mathcal{O} restricted to the open intervals in the time partition requires a new technique. We can however benefit from the fact that throughout each interval, all snapshots of \mathcal{O} have an \mathcal{T}_S -isomorphic triangulation. For each of the open intervals defined by two subsequent elements $]\tau_j, \tau_{(j+1)}[$ of χ , we compute the snapshot at the middle $\tau_m = \frac{1}{2}(\tau_j + \tau_{(j+1)})$ of $]\tau_j, \tau_{(j+1)}[$ and its triangulation $\mathcal{T}_S(\mathcal{O}^{\tau_m})$. Each triangle boundary segment that contributes to the boundary of \mathcal{O}^{τ_m} at time moment τ_m , will also contribute to the boundary of \mathcal{O} at the snapshot of \mathcal{O} at any time moment $\tau \in]\tau_j, \tau_{(j+1)}[$. So, the moving line segment can be considered a boundary segment throughout $]\tau_j, \tau_{(j+1)}[$. If two carriers of boundary segments intersect at time moment τ_m , the intersection of the moving segments will exist throughout $]\tau_j, \tau_{(j+1)}[$, and so on. Therefor, we will compute the spatial triangulation of the snapshot \mathcal{O}^{τ_m} using the procedure \mathcal{T}_S , but we will copy every action on a point or line segment at time moment τ_m on the moving point of line segment of which the point or segment is a snapshot. The triangles returned by the spatial triangulation algorithm when applied to \mathcal{O}^{τ_m} will be reference objects for the atomic objects, returned by the spatio-temporal triangulation algorithm. These atomic objects exist during the interval $]\tau_j, \tau_{(j+1)}[$. Knowing the functions representing the time-dependent corner points of the triangles (because of the copying), together with the time interval and the reference object, we can deduce the transformation function and construct atomic objects (the formula computing this transformation was given in [5]).

Next, a detailed description of the spatio-temporal triangulation is given in Algorithm 3. In this description of the spatio-temporal triangulation procedure, we will use the data type *Points* which is a structure containing a (2-dimensional) point (represented using a pair of real numbers), a pair of rational functions of t (a rational function is represented using a pair of vectors of integers, denoting the coefficients of a polynomial), representing a moving point, and finally a time interval (represented as a pair of real numbers and two flags indicating whether the interval is open or closed at each end point). We will only use or fill in this time information when mentioned explicitly. Given an element *Pt* of type *Points*, we address the point it stores by $Pt \rightarrow Point$, the functions of time by $Pt \rightarrow f_x$ and $Pt \rightarrow f_y$ respectively, and the begin and end point of the time interval by $Pt \rightarrow I_b$ and $Pt \rightarrow I_e$. The flags $Pt \rightarrow C_b$ and $Pt \rightarrow C_e$ are true when the interval is closed at its begin or end point respectively. A pair of elements of the type *Points* is denoted an element of the type *Segments*.

We again illustrate the spatio-temporal triangulation algorithm on the geometric object of example 8.

Example 10 Recall from Example 8 that $\mathcal{O} = \{\mathcal{O}_1, \mathcal{O}_2\}$, where \mathcal{O}_1 is given as $(((-1, 0), (1, 0), (0, 2)), [0, 4], Id)$ and \mathcal{O}_2 is given as $(((-3, 1), (-1, 1), (-2, 3)), [0, 4], f)$ and f is the affinity mapping triples (x, y, t) to pairs $(x + t, y)$.

Algorithm 3 Triangulate (Input: $\mathcal{O} = \{\mathcal{O}_1, \dots, \mathcal{O}_n\}, \chi = (\tau_1, \dots, \tau_k)$, Output = $\{\mathcal{O}'_1, \dots, \mathcal{O}'_\ell\}$)

```

1: for all time moments  $\tau_j, j = 1 \dots k$ , of  $\chi$  do
2:   for all triangles  $T$  in  $\mathcal{T}_S(\mathcal{O}^{\tau_j})$  do
3:     return the atomic element  $(T, \{\tau_j\}, Id)$ .
4:   end for
5: end for
6: Let  $S_{<}$  be the list containing all atomic objects  $\mathcal{O}_i = (S_i, I_i, f_i), 1 \leq i \leq n$ ,
   sorted by the begin points  $I_{i,b}$  of their time domains.
7: Let  $S_{\text{Active}}$  be a list of elements of the type Segments,  $S_{\text{Active}} = ()$ .
8: for all pairs  $(\tau_j, \tau_{j+1}), j = 1 \dots (k - 1)$ , in  $\chi$  do
9:    $\tau_m := \frac{1}{2}(\tau_j + \tau_{j+1})$ .
10:  Remove all elements  $(Pt_1, Pt_2)$  of  $S_{\text{Active}}$  for which  $\tau_j = Pt_1 \rightarrow I_e =$ 
    $Pt_2 \rightarrow I_e$ .
11:  for all elements  $(Pt_1, Pt_2)$  remaining in  $S_{\text{Active}}$  do
12:     $Pt_r \rightarrow Point := (Pt_r \rightarrow f_x(\tau_m), Pt_r \rightarrow f_y(\tau_m)), r = 1, 2$ .
13:  end for
14:  for all  $\mathcal{O}_i = (S_i = (\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3), I_i, f_i)$  in  $S_{<}$  for which  $I_{i,b}$  is  $\tau_i$  do
15:    Construct three Points  $Pt_1, Pt_2$  and  $Pt_3$  such that  $Pt_r \rightarrow Point =$ 
    $\mathbf{a}_r, Pt_r \rightarrow f_x = f_i(a_{r,x}, \tau_m), Pt_r \rightarrow f_y = f_i(a_{r,y}, \tau_m)$  and  $Pt_r \rightarrow I_b$ 
   and  $Pt_r \rightarrow I_e$  respectively contain  $\tau_j$  and  $\tau_{j+1}$  ( $r = 1, \dots, 3$ ).
16:    Construct three Segments  $St_1, St_2$  and  $St_3$ , containing two different
   elements from the set  $\{Pt_1, Pt_2, Pt_3\}$ . Add them to  $S_{\text{Active}}$ .
17:  end for
18:  Compute the set  $\mathcal{B}^t(S_{\text{Active}})$  of elements of the type Segments, using
   only the constant point information of the elements of  $S_{\text{Active}}$ . Mean-
   while, construct the subdivision  $\mathcal{U}(\mathcal{O}^{\tau_m})$ .
19:  Compute the convex hull  $\mathcal{CH}^t(S_{\text{Active}})$ , using only the constant point
   information of the elements of  $S_{\text{Active}}$ , a list of elements of the type
   Points.
20:  Construct  $\text{DCEL}^t(S_{\text{Active}})$ , where each half-edge (resp. origin) is now
   an element of the type Segments (resp. Points). Use  $\mathcal{CH}^t(S_{\text{Active}})$  as
   a bounding box. Each time the intersection of two constant carriers
   is computed, also compute the formula representing the moving inter-
   section point.
21:  while there are any unvisited Segments  $St$  in  $\text{DCEL}^t(S)$  left do
22:    Compute the list Etlist of Segments that form a convex polygon.
    Compute the Points structure  $Pt_m$  containing both the constant
    and time-dependent center of mass of that polygon
23:    if  $Pt_m \rightarrow Point$  belongs to a face of  $\mathcal{U}(\mathcal{O}^{\tau_m})$  then
24:      for all elements  $St = (Pt_1, Pt_2)$  of Etlist do
25:        Output the atomic object  $(S, I, f)$ , where  $S$  is the triangle with
        corner points  $Pt_1 \rightarrow Point, Pt_2 \rightarrow Point$  and  $Pt_m \rightarrow Point$ 
        and  $I$  is  $] \tau_j, \tau_{j+1} [$ . The transformation function  $f$  is computed
        using the functions  $Pt_1 \rightarrow f_x, Pt_1 \rightarrow f_y, Pt_2 \rightarrow f_x, Pt_2 \rightarrow f_y,$ 
 $Pt_m \rightarrow f_x$  and  $Pt_m \rightarrow f_y$ 
26:      end for
27:    end if
28:  end while

```

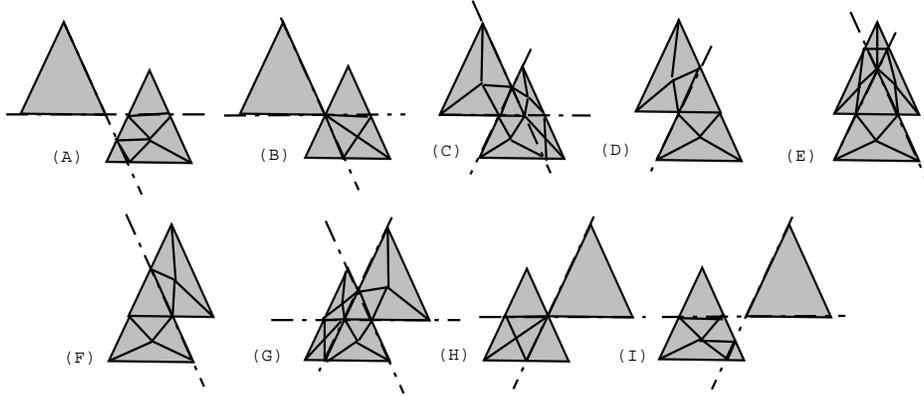


Figure 9: The triangulations of the objects of Example 8 at time moments $t = \frac{1}{4}$ (A), $t = \frac{1}{2}$ (B), $t = 1$ (C), $t = \frac{3}{2}$ (D), $t = 2$ (E), $t = \frac{5}{2}$ (F), $t = 3$ (G), $t = \frac{7}{2}$ (H) and $t = 4$ (I).

From Example 9, we recall that the output of the procedure **Partition** on input \mathcal{O} was the list $\chi = (0, \frac{1}{2}, \frac{3}{2}, \frac{5}{2}, \frac{7}{2}, 4)$.

The triangulation of the snapshots at one of the time moments in χ are shown in Figure 9. To keep the example as simple as possible, we did not further triangulate convex polygons that are triangles already.

The open intervals to be considered are $]0, \frac{1}{2}[$, $]\frac{1}{2}, \frac{3}{2}[$, $]\frac{3}{2}, \frac{5}{2}[$, $]\frac{5}{2}, \frac{7}{2}[$ and $]\frac{7}{2}, 4[$. We illustrate the triangulation of the interval $]0, \frac{1}{2}[$. During the time interval $]0, \frac{1}{2}[$, the triangulation will always look like the one shown in Part (A) of Figure 9. Hence, \mathcal{O}_2 will not change, and \mathcal{O}_1 will be partitioned into seven triangles. The top one will not change, so the atomic object $((0, 2), (1, \frac{-1}{2}), (1, \frac{1}{2}),]0, \frac{1}{2}[, Id)$ will be part of the output. For the others, we have to compute the time-dependent intersections between the carriers and afterwards apply the formula from [5]. We illustrate this for \mathcal{O}_2 . the snapshot of \mathcal{O}_2 at the middle point $\frac{1}{4}$ of $]0, \frac{1}{2}[$ is the triangle with corner points $(\frac{-11}{4}, 1)$, $(\frac{-3}{4}, 1)$ and $(\frac{-7}{4}, 3)$. Its time-dependent corner points are $(-3+t, 1)$, $(-1+t, 1)$ and $(-2+t, 3)$. Solving the matrix equation

$$\begin{pmatrix} \frac{-11}{4} & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{-11}{4} & 1 & 0 & 1 \\ \frac{-7}{4} & 3 & 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{-7}{4} & 3 & 0 & 1 \\ \frac{-3}{4} & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{-3}{4} & 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} a(t) \\ b(t) \\ c(t) \\ d(t) \\ e(t) \\ f(t) \end{pmatrix} = \begin{pmatrix} -3+t \\ 1 \\ -2+t \\ 3 \\ -1+t \\ 1 \end{pmatrix}$$

gives the transformation function f' that maps triples (x, y, t) to pairs $(x - \frac{1}{4} + t, y)$. \square

We also give the output complexity and time complexity for this triangulation step.

Property 8 (Triangulation step: output complexity) Given a geometric object $\mathcal{O} = \{\mathcal{O}_1 = (S_1, I_1, f_1), \mathcal{O}_2 = (S_2, I_2, f_2), \dots, \mathcal{O}_n = (S_n, I_n, f_n)\}$ consisting of n atomic objects and a finite partition χ of its time domain into k time points and $k - 1$ open intervals. The procedure **Triangulation**, as described in Algorithm 3, returns $O(n^2k)$ atomic objects.

Proof. The number of atomic objects returned by the triangulation procedure for one time interval is the same as the number of triangles returned by the spatial triangulation method on a snapshot in that interval. We know from Property 2 that the number of triangles in the triangulation of a snapshot composed from n triangles is $O(n^2)$. Since there are $O(k)$ moments and intervals for which we have to consider such a triangulation, or a slightly adapted version of it, this gives $O(n^2k)$. \square

Property 9 (Triangulation step: computational complexity) Let a geometric object $\mathcal{O} = \{\mathcal{O}_1 = (S_1, I_1, f_1), \mathcal{O}_2 = (S_2, I_2, f_2), \dots, \mathcal{O}_n = (S_n, I_n, f_n)\}$ consisting of n atomic objects and a finite partition χ of its time domain into k time points and $k - 1$ open intervals be given. Let d be the maximal degree of any polynomial in the definition of the transformation functions $f_i, 1 \leq i \leq n$ and let ϵ be the desired precision for computing the zeros of polynomials. The procedure **Partition**, as described in Algorithm 2, returns a spatio-temporal triangulation of \mathcal{O} in time $O(kz(d, \epsilon)n^2 \log n)$.

Proof. The first for-loop of Algorithm 3 is executed k times. The time needed for computing the snapshot of one atomic object at a certain time moment is $z(d, \epsilon)$. The spatial triangulation algorithm \mathcal{T}_S runs in time $O(n^2 \log n)$, as was shown in Property 3. So we can conclude that the body of the first for-loop needs $O(n^2 \log n + nz(d, \epsilon))$ time. Sorting the atomic objects by their time domains takes $O(n \log n)$.

The second for loop is executed once for each open interval, defined by two consecutive elements of χ . In the body of this loop, first the list S_{Active} is updated. Each insertion or update takes time $z(d, \epsilon)$. At most all objects are in the list S_{Active} , so this part, described in the Lines 10 through 18 of Algorithm 3, needs time $O(nz(d, \epsilon))$. The next part, described in the Lines 19 through 29 essentially is the spatial triangulation algorithm, but, any time the intersection between two line segments is computed, also the rational functions defining the time-dependent intersection of their associated time-dependent line segments are computed. Computing those functions takes time $z(d, \epsilon)$. So the second part of the body of the second for loop requires $O(z(d, \epsilon)n^2 \log n)$.

If we add up the time complexity of two for-loops and the sorting step, we have $O(k(nz(d, \epsilon) + n^2 \log n) + n \log n + kz(d, \epsilon)(n + n^2 \log n))$, which is $O(kz(d, \epsilon)n^2 \log n)$. \square

4.3 The Merge Step.

We already mentioned briefly in the description of the partitioning step that the partition of the time domain, as computed by Algorithm 2, might be finer than necessary. The partitioning algorithm takes into account all line segments, also those of objects that, during some time span, are entirely overlapped by other objects. To solve this, we merge as much elements of the time partition as possible.

The partition of the time domain is such that the merging algorithm will either try to merge a time point τ and an interval of the type $]\tau, \tau')$ or $(\tau', \tau[$, or two different intervals of the type $(\tau'', \tau[$ and $]\tau, \tau')$ or $(\tau'', \tau[$ and $[\tau, \tau')$. Here, we use the (unusual) notational convention that $($ and $)$ can be either $[$ or $]$.

The simplest case is when a time moment and an interval have to be tested. Assume that these are $(\tau', \tau[$ and τ , respectively. These elements can be merged if there is a one to one mapping M from the atomic elements with time domain $(\tau', \tau[$ to those with time domain $\{\tau\}$ in the triangulation. Furthermore, for each pair of atomic objects $\mathcal{O}_1 = (S_1, (\tau', \tau[, f_1)$ and $\mathcal{O}_2 = (S_2, \{\tau\}, Id)$, $\mathcal{O}_2 = M(\mathcal{O}_1)$ if and only if the left limit $\lim_{t \rightarrow \tau} f_1(S_1, t) = S_2$. Note that, for rational functions f of t , $\lim_{t \rightarrow \tau} f(t)$ equals $f(\tau)$, provided that τ is in the domain of f ⁴.

When two intervals are to be merged, the procedure involves some more tests. Let $(\tau'', \tau[$ and $[\tau, \tau')$ be the intervals to be tested. First, we have to verify that for each atomic object $\mathcal{O}_1 = (S_1, (\tau'', \tau[, f_1)$, $[\tau, \tau')$ is in the domain of f_1 and that for each atomic object $\mathcal{O}_2 = (S_2, [\tau, \tau'), f_2)$, $(\tau'', \tau[$ is in the domain of f_2 . Second, we have to test whether $(\tau'', \tau[$ can be continuously expanded to $(\tau'', \tau]$. This involves the same tests as for the simple case where an interval and a point are tested. Finally, two atomic object can only be merged if the combined atomic object again is an atomic object. This means that, if S_2 would have been chosen as a reference object for \mathcal{O}_1 , then f_1 would be equal to f_2 , and vice versa. This can be tested ([5]).

This merge step guarantees that the atomic objects exist maximally and that the resulting triangulation is the same for geometric objects that represent the same spatio-temporal object. Algorithm 4 shows this merging step in detail.

We illustrate Algorithm 4 on the geometric object of Example 8.

Example 11 Recall from Example 8 that $\mathcal{O} = \{\mathcal{O}_1, \mathcal{O}_2\}$, where \mathcal{O}_1 is given as $(((-1, 0), (1, 0), (0, 2)), [0, 4], Id)$ and \mathcal{O}_2 is given as $(((-3, 1), (-1, 1), (-2, 3)), [0, 4], f)$ and f is the linear affinity mapping triples (x, y, t) to pairs $(x + t, y)$.

⁴Note that τ is in the domain of f only if all coefficients of the transformation function f are well-defined for $t = \tau$ and if the determinant of f is nonzero for $t = \tau$.

Algorithm 4 Merge (Input: $\mathcal{O} = \{\mathcal{O}_1, \dots, \mathcal{O}_n\}$, $\chi = (\tau_1, \tau_2, \dots, \tau_k)$, Output: $\{\mathcal{O}'_1, \mathcal{O}'_2, \dots, \mathcal{O}'_\ell\}$)

```

1: Sort all atomic objects  $\mathcal{O}_i$  by their time domains.
2: Let  $\chi'$  be the list  $(\tau_1, ]\tau_1, \tau_2[, \tau_2, \dots, ]\tau_{k-1}, \tau_k[, \tau_k)$ .
3: Let  $J_1$  be the first element of  $\chi'$  and  $J_2$  the second.
4: while there are any elements in  $\chi'$  left do
5:    $\mathcal{S}_1$  (resp.  $\mathcal{S}_2$ ) is the set of all objects having  $J_1$  (resp.  $J_2$ ) as their time domain.
6:   if  $J_1$  is a point then
7:     Preprocess the reference objects of the elements of  $\mathcal{S}_1$  such that we can search the planar subdivision  $\mathcal{U}_1$  they define.
8:     let Found be true.
9:     for all objects  $\mathcal{O}_i = (S_i, J_2, f_i)$  in  $\mathcal{S}_2$  do
10:      Check whether  $J_1$  is part of the time domain of  $f_i$ .
11:      Compute their snapshot at time  $J_1$  (which is a triangle  $T$ ).
12:      Do a point location query with the center of mass of  $T$  in  $\mathcal{U}_1$  and check whether the triangle found in  $\mathcal{S}_1$  has the same coordinates as  $T$ . If not, Found becomes false.
13:      if Found is false then
14:        break;
15:      end if
16:    end for
17:    if found is true then
18:      remove all elements of  $\mathcal{S}_1$  from  $\mathcal{O}$  and extend the time domain of all elements of  $\mathcal{S}_2$  to  $J_1 \cup J_2$ .
19:       $J_1 = J_1 \cup J_2$  and  $J_2$  is the next element of  $\chi'$  if any exists.
20:    else
21:       $J_1 = J_2$  and  $J_2$  is the next element of  $\chi'$ , if any exists.
22:    end if
23:  else
24:    if  $J_2$  is a point then
25:      do the same as in the previous case, but switch the roles of  $J_1$  and  $J_2$ .
26:    else
27:      Let  $J'_1$  be the element of  $\{J_1, J_2\}$  the form  $(\tau'', \tau[$  and  $J'_2$  the one of the form  $[\tau, \tau')$ .
28:      Check whether  $(\tau'', \tau[$  and  $\{\tau\}$  can be merged.
29:      if this can be done then
30:        Check for each pair of matching atomic objects whether their transformation functions are the same ([5]).
31:      end if
32:    end if
33:  end if
34: end while

```

From Example 9, we recall that the output of the procedure **partition** on input \mathcal{O} was the list $\chi = (0, \frac{1}{2}, \frac{3}{2}, \frac{5}{2}, \frac{7}{2}, 4)$. This resulted in a partition of the interval $[0, 4]$ consisting of the elements $\{0\}$, $]0, \frac{1}{2}[$, $\{\frac{1}{2}\}$, $] \frac{1}{2}, \frac{3}{2}[$, $\{\frac{3}{2}\}$, $] \frac{3}{2}, \frac{5}{2}[$, $\{\frac{5}{2}\}$, $] \frac{5}{2}, \frac{7}{2}[$, $\{\frac{7}{2}\}$, $] \frac{7}{2}, 4[$ and $\{4\}$. For each of these elements, (a snapshot of) their triangulation is shown in Figure 9.

During the merge step, the elements $t = 0$ and $]0, \frac{1}{2}[$ of the time partition will be merged. \square

It is straightforward that the output and input of the merging algorithm have the same order of magnitude. Indeed, it is possible that no intervals are merged, and hence no objects. We discuss the computational complexity of the algorithm next. Note that the complexity is expressed in terms of the size of the input to the merging algorithm, which is the output of the spatio-temporal triangulation step.

Property 10 (Merge step: computational complexity) Given a geometric object $\mathcal{O} = \{\mathcal{O}_1 = (S_1, I_1, f_1), \mathcal{O}_2 = (S_2, I_2, f_2), \dots, \mathcal{O}_n = (S_n, I_n, f_n)\}$, which is the output of the triangulation step, and a finite partition χ of its time domain into K time moments and open intervals. Let d be the maximal degree of any polynomial in the definition of the transformation functions $f_i (1 \leq i \leq n)$ and let ϵ be the desired precision for computing the zeros of polynomials. The procedure **Merge**, as described in Algorithm 4, merges the atomic objects in \mathcal{O} in time $O(n \log \frac{n}{K} + nz(d, \epsilon))$.

Proof. Sorting all atomic objects by their time domains can be done in time $O(n \log n)$. Computing the list χ' can be straightforwardly done in time $O(K)$. This list will contain $2K - 1$ elements. We assume that $K > 1$ (in case $K = 1$ the merging algorithm is not applied). The while-loop starting at Line 4 of Algorithm 4, is executed at most $2K - 2$ times. Indeed, at each execution of the body of the while-loop, one new element of χ' is considered. The **if-else** structure in the body of the while-loop distinguishes three cases. All cases have the same time complexity, as they are analogous. We explain the first case in detail.

The number of atomic objects having the same time domain is of the order of magnitude of $O(\frac{n}{K})$. This follows from Property 8. The preprocessing of the snapshot takes $O(\frac{n}{K})$ time [13]. The for-loop, starting at Line 9 of Algorithm 4 is executed at most $O(\frac{n}{K})$ times. The time needed for checking whether an atomic object exists at some time moment and computing the snapshot (a triangle) is $z(d, \epsilon)$. Because of the preprocessing on the snapshot at time moment J_1 , testing the barycenter of the triangle against that snapshot can be done in $O(\log \frac{n}{K})$ time [13]. In case the snapshots are the same, adjusting the time domains of all atomic objects takes time $O(\frac{n}{K})$. Summarizing, the time complexity of the first case is $O(\frac{n}{K} \log \frac{n}{K} + \frac{n}{K} z(d, \epsilon))$.

Step	Time complexity	Output complexity
Partition	$O(z(d, \epsilon)n^3 \log n)$	$O(n^3d)$
Triangulate	$O(z(d, \epsilon)dn^5 \log n)$	$O(n^5d)$
Merge	$O(n^5d(\log n + z(d, \epsilon)))$	-
Overall	$O(z(d, \epsilon)dn^5 \log n)$	$O(n^5d)$

Table 5: The output and time complexity of the various parts of Algorithm 5, when the input is a geometric object, composed of n atomic objects, where the maximal degree of the polynomials describing the transformation functions is d and the desired precision for computing the zeros of polynomials is ϵ .

Combining this with the fact that the while-loop is executed $O(K)$ times, and the time complexity of the first two steps of the algorithm, we get an overall time complexity of $O(n \log \frac{n}{K} + nz(d, \epsilon))$. \square

Finally, the spatio-temporal triangulation procedure \mathcal{T}_{ST} combines the partition, triangulation and merging step. Algorithm 5 combines all steps.

Algorithm 5 \mathcal{T}_{ST} (Input: $\mathcal{O} = \{\mathcal{O}_1, \dots, \mathcal{O}_n\}$, Output: $\{\mathcal{O}'_1, \mathcal{O}'_2, \dots, \mathcal{O}'_\ell\}$)

- 1: $\chi = \mathbf{Partition}(\mathcal{O})$;
 - 2: $\{\mathcal{O}''_1, \mathcal{O}''_2, \dots, \mathcal{O}''_m\} = \mathbf{Triangulate}(\mathcal{O}, \chi)$;
 - 3: **if** χ has more than one element **then**
 - 4: $\{\mathcal{O}'_1, \mathcal{O}'_2, \dots, \mathcal{O}'_\ell\} = \mathbf{Merge}(\{\mathcal{O}''_1, \mathcal{O}''_2, \dots, \mathcal{O}''_m\}, \chi)$;
 - 5: return $\{\mathcal{O}'_1, \mathcal{O}'_2, \dots, \mathcal{O}'_\ell\}$.
 - 6: **else**
 - 7: return $\{\mathcal{O}''_1, \mathcal{O}''_2, \dots, \mathcal{O}''_m\}$.
 - 8: **end if**
-

The following property follows from Property 6 and Property 8.

Property 11 (\mathcal{T}_{ST} : output complexity) Given a geometric object $\mathcal{O} = \{\mathcal{O}_1 = (S_1, I_1, f_1), \mathcal{O}_2 = (S_2, I_2, f_2), \dots, \mathcal{O}_n = (S_n, I_n, f_n)\}$ consisting of n atomic objects. Let d be the maximal degree of any polynomial in the definition of the transformation functions $f_i (1 \leq i \leq n)$. The spatio-temporal triangulation method \mathcal{T}_{ST} , as described in Algorithm 5, returns $O(n^5d)$ atomic objects.

The next property follows from Property 7, Property 9 and Property 10. Table 5 summarizes the time complexity of the different steps.

Property 12 (\mathcal{T}_{ST} : computational complexity) Given a geometric object $\mathcal{O} = \{\mathcal{O}_1 = (S_1, I_1, f_1), \mathcal{O}_2 = (S_2, I_2, f_2), \dots, \mathcal{O}_n = (S_n, I_n, f_n)\}$ consisting of n atomic objects. Let d be the maximal degree of any polynomial

in the definition of the transformation functions $f_i(1 \leq i \leq n)$ and let ϵ be the desired precision for computing the zeros of polynomials. The spatio-temporal triangulation method \mathcal{T}_{ST} , as described in Algorithm 5, returns a spatio-temporal triangulation of \mathcal{O} in time $O(z(d, \epsilon)dn^5 \log n)$.

We now show that Algorithm 5 describes an affine-invariant spatio-temporal triangulation method. We remark first that the result of the procedure \mathcal{T}_{ST} is a spatio-temporal triangulation. Given a geometric object \mathcal{O} . It is clear that each snapshot of $\mathcal{T}_{ST}(\mathcal{O})$ is a spatial triangulation. Also, $st(\mathcal{O}) = st(\mathcal{T}_{ST}(\mathcal{O}))$. This follows from the fact that the time partition covers the whole time domain of \mathcal{O} and that the method \mathcal{T}_S produces a spatial triangulation.

Property 13 (\mathcal{T}_{ST} is affine-invariant) The spatio-temporal triangulation method \mathcal{T}_{ST} , described in Algorithm 5, is affine-invariant.

Proof. (Recall Definition 6 for affine-invariance.) Let $\mathcal{O} = \{\mathcal{O}_1, \dots, \mathcal{O}_n\}$ and $\mathcal{O}' = \{\mathcal{O}'_1, \dots, \mathcal{O}'_m\}$ be geometric objects for which for each moment τ_0 in their time domains, there is an affinity $\alpha_{\tau_0} : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ such that $\alpha_{\tau_0}(\{\mathcal{O}_1, \dots, \mathcal{O}_n\}^{\tau_0}) = \{\mathcal{O}'_1, \dots, \mathcal{O}'_m\}^{\tau_0}$.

It follows from the construction of the spatio-temporal triangulation that $\mathcal{T}_{ST}(\{\mathcal{O}_1, \dots, \mathcal{O}_n\})^{\tau_0} = \mathcal{T}_S(\{\mathcal{O}_1, \dots, \mathcal{O}_n\}^{\tau_0})$ and also $\mathcal{T}_{ST}(\{\mathcal{O}'_1, \dots, \mathcal{O}'_m\})^{\tau_0} = \mathcal{T}_S(\{\mathcal{O}'_1, \dots, \mathcal{O}'_m\}^{\tau_0})$. The property now follows from the affine-invariance of the spatial triangulation method \mathcal{T}_S . \square

The following corollary follows straightforwardly from Property 13:

Corollary 1 Let $\mathcal{O} = \{\mathcal{O}_1, \dots, \mathcal{O}_n\}$ and $\mathcal{O}' = \{\mathcal{O}'_1, \dots, \mathcal{O}'_m\}$ be two geometric objects such that there is an affinity $\alpha : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ such that, for each moment τ_0 in their time domains $\alpha(\mathcal{O}^{\tau_0}) = \mathcal{O}'^{\tau_0}$ holds. Then, for each atomic element (S, I, f) of $\mathcal{T}_{ST}(\mathcal{O})$, the element $(\alpha(S), I, f)$ belongs to $\mathcal{T}_{ST}(\mathcal{O}')$. \square

This shows that the partition is independent of the coordinate system used to represent the spatio-temporal object. The affine partitions of two spatio-temporal objects that are affine images of each other only differ in the coordinates of the spatial reference objects of the atomic objects.

Remark also that, in practice, either most of the original objects have the same time domain, making the number of intervals in the partition very small, or all different time domains, which greatly reduces the number of objects existing during each interval. So, in practice, the performance will be better than the worst case suggests.

5 Applications

We now describe some applications that we believe can benefit from the triangulation described in Algorithm 5. We first say what we mean by a spatio-temporal database.

Definition 11 A *spatio-temporal database* is a set of geometric objects. \square

For this section, we assume that each atomic object is labelled with the id of the geometric object it belongs to.

5.1 Efficient Rendering of Objects

When a geometric object that is not in normal form has to be displayed to the user, there are two tasks to perform. First, the snapshots of the geometric object at each time moment in the time domain of the object have to be computed. This can be done in a brute force way by computing the snapshots of all atomic objects. Since some will be empty, this approach might lead to a lot of unnecessary computations. Another algorithm could keep track of the time domains of the individual atomic objects and keep a list of *active* ones at the moment under consideration, which has to be updated every instant. If the geometric object is in normal form, the atomic objects can be sorted by their time domains, and during each interval in the partition of time domain, the list of active atomic objects will remain the same.

The second task is the rendering of the snapshot. If the geometric object is not in normal form, the snapshots of the atomic objects overlap, so pixels will be computed more than once. Another solution is computing the boundary, but this might take too long in real-time applications. When a geometric object is in normal form, no triangles overlap, so each pixel will be computed only once.

5.2 Moving Object Retrieval

The triangulation provides a means of automatic affine invariant feature extraction for moving object recognition. Indeed, the number of intervals in the time domain indicates the complexity of the movement of the geometric object. This can be used as a first criterium for object matching. For objects having approximately the same number of intervals in their time domains, the snapshots at the middle of each time interval can be compared. If they are all similar, which can be, for example, defined as \mathcal{T}_S -isomorphic, the objects match. Or, if more exact comparison is needed, one can extract an affine-invariant description from the structure of the elements of the triangulation of the snapshot (see also Section 5.6).

5.3 Surveillance Systems

In some applications, e.g., surveillance systems, it is important to know the time moments when something changed, when some discontinuity appeared. This could mean that an unauthorized person entered a restricted area, for

example, or that a river has burst its banks. Triangulating the contours of the recorded images and reporting all single points and end points of intervals of the partition of the time domain indicates all moments when some discontinuity might have occurred.

5.4 Precomputing Queries

If we do not triangulate each geometric object in a database separately, but use the contours of all geometric objects together in the triangulation, the atomic objects in the result will have the following nice property. For each geometric object in the original database, an atomic object will either belong to (or be a subset of) it entirely, or not at all. This means that we can label each element of the spatio-temporal triangulation of the database with the set of id's of the geometric objects it belongs to. We illustrate this for the spatial case only in Figure 10. Suppose we have two triangles A and B . The set A is the union of the light grey and white parts of the figure, the set B is the union of the dark grey and white parts of the figure. After triangulation, we can label the light grey triangles with $\{A\}$, the white triangle with $\{A, B\}$, and the dark grey triangles with $\{B\}$.

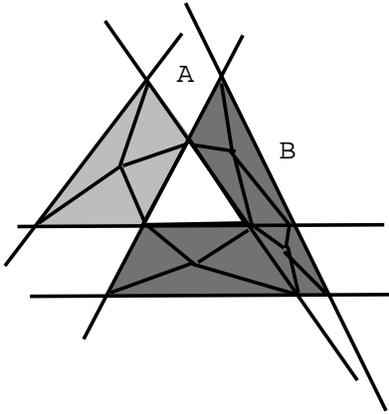


Figure 10: The set operations between objects are pre-computed in the triangulation.

Using this triangulation of databases in a preprocessing stage, means that the results of queries that ask for set operations between geometric objects are also pre-computed. Answering such a query boils down to checking labels of atomic objects. This means a lot of gain in speed at query time. Indeed, even to compute, for example, the intersection of two atomic objects, one has to compute first the intervals where the intersection exists, and then to consider all possible shapes the intersection can have, and again represent it by moving triangles.

5.5 Maintaining the Triangulation.

If a geometric object has to be inserted into or removed from a database (i.e., a collection of geometric objects), the triangulation has to be recomputed for the intervals in the partition of the time domain that contain the time domain of the object under consideration. This may require that the triangulation in total has to be recomputed.

However, the nature of a lot of spatio-temporal applications is such that updates involve only the insertion of objects that exists *later* in time than the already present data. In that case, only the triangulation at the latest time interval of the partition should be recomputed together with the new object, to check whether the new data are a continuation of the previous. Also, data is removed only when it is outdated. In that case a whole time interval of data can be removed. Examples of such spatio-temporal applications are surveillance, traffic monitoring and cadastral information systems.

5.6 Affine-invariant Querying of Spatio-temporal Databases

An interesting topic for further work is to compute a new, affine-invariant description of geometric objects in normal form, that does not involve coordinates of reference objects. The structure of the atomic objects in the spatio-temporal triangulation can be used for that. Once such a description is developed, a query language can be designed that asks for affine-invariant properties of objects only.

6 Concluding Remarks

We adopted the hierarchical data model of Chomicki and Revesz [6] for moving objects, since it is natural and flexible. However, this model lacks a normal form, as different sets of objects might represent the same spatio-temporal set in $\mathbb{R}^2 \times \mathbb{R}$. Furthermore, we are interested in affine-invariant representation and querying of objects, as the choice of origin and unit of measure should not affect queries on spatio-temporal data.

We first introduced a new affine-invariant triangulation method for spatial data. We then extended this method for spatio-temporal data in such a way that the time domain is partitioned in intervals for which the triangulations of all snapshots are *isomorphic*.

The proposed affine-invariant triangulation is natural and can serve as an affine-invariant normal form for spatio-temporal data. Further work includes the affine-invariant finite representation of data and the design of an affine-generic spatial/ spatio-temporal query language to query such a normal form.

References

- [1] *Proceedings of the 7th International Workshop on Temporal Representation and Reasoning*. IEEE Computer Society Press, 2000.
- [2] J. Bochnak, M. Coste, and M.-F. Roy. *Real Algebraic Geometry*, volume 36 of *Ergebnisse der Mathematik und ihrer Grenzgebiete. Folge 3*. Springer-Verlag, 1998.
- [3] M. H. Böhlen, Ch. S. Jensen, and M. Scholl, editors. *Spatio-Temporal Database Management, International Workshop STDBM 1999*, volume 1678 of *Lecture Notes in Computer Science*. Springer, 1999.
- [4] C. X. Chen and C. Zaniolo. SQLST: A spatio-temporal data model and query language. In V. C. Storey A. H. F. Laender, S. W. Liddle, editor, *Conceptual Modeling, 19th International Conference on Conceptual Modeling (ER'00)*, volume 1920 of *Lecture Notes in Computer Science*, pages 96–111. Springer-Verlag, 2000.
- [5] J. Chomicki, S. Haesevoets, B. Kuijpers, and P. Revesz. Classes of spatiotemporal objects and their closure properties. *Annals of Mathematics and Artificial Intelligence*, (39):431–461, 2003.
- [6] J. Chomicki and P. Revesz. A geometric framework for specifying spatiotemporal objects. In *Proceedings of the 6th International Workshop on Temporal Representation and Reasoning*, pages 41–46. IEEE Computer Society Press, 1999.
- [7] G.E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In H. Brakhage, editor, *Automata Theory and Formal Languages*, volume 33 of *Lecture Notes in Computer Science*, pages 134–183, Berlin, 1975. Springer-Verlag.
- [8] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. Arrangements and duality. In *Computational Geometry: Algorithms and Applications*, chapter 8, pages 165–182. Springer-Verlag, 2000.
- [9] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. Computational geometry. In *Computational Geometry: Algorithms and Applications*, chapter 1, pages 1–17. Springer-Verlag, 2000.
- [10] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 2000.
- [11] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. Line segment intersection. In *Computational Geometry: Algorithms and Applications*, chapter 2, pages 18–43. Springer-Verlag, 2000.

- [12] F. Dumortier, M. Gyssens, L. Vandeurzen, and D. Van Gucht. On the decidability of semi-linearity for semi-algebraic sets and its implications for spatial databases. In *Proceedings of the 16th ACM Symposium on Principles of Database Systems*, pages 68–77. ACM Press, 1997.
- [13] H. Edelsbrunner, L. J. Guibas, and J. Stolfi. Optimal point location in a monotone subdivision. *SIAM J. Comput.*, 15(2):317–340, 1986.
- [14] M. Erwig and M. Schneider. Partition and conquer. In *Proceedings of the 3rd International Conference on Spatial Information Theory*, volume 1329 of *Lecture Notes in Computer Science*, pages 389–408. Springer, 1997.
- [15] A. Frank, S. Grumbach, R. Güting, C. Jensen, M. Koubarakis, N. Lorentzos, Y. Manopoulos, E. Nardelli, B. Pernici, H.-J. Schek, M. Scholl, T. Sellis, B. Theodoulidis, and P. Widmayer. Chorochronos: A research network for spatiotemporal database systems. *SIGMOD Record*, 28:12–21, 1999.
- [16] F. Geerts, S. Haesevoets, and B. Kuijpers. A theory of spatio-temporal database queries. In G. Ghelli and G. Grahne, editors, *Database Programming Languages, 8th International Workshop, DBPL 2001*, volume 2397 of *Lecture Notes in Computer Science*, pages 198–212. Springer, 2002.
- [17] S. Grumbach, P. Rigaux, and L. Segoufin. Spatio-temporal data handling with constraints. In R. Laurini, K. Makki, and N. Pissinou, editors, *Proceedings of the 6th International Symposium on Advances in Geographic Information Systems (ACM-GIS'98)*, pages 106–111, 1998.
- [18] R. H. Güting, M. H. Bohlen, M. Erwig, C. S. Jensen, N. A. Lorentzos, M. Schneider, and M. Vazirgiannis. A foundation for representing and querying moving objects. *ACM Transactions on Databases Systems*, 25:1–42, 2000.
- [19] M. Gyssens, J. Van den Bussche, and D. Van Gucht. Complete geometric query languages. *Journal of Computer and System Sciences*, 58(3):483–511, 1999.
- [20] S. Haesevoets and B. Kuijpers. Time-dependent affine triangulation of spatio-temporal data. In *12th ACM International Workshop on Geographic Information Systems, ACM-GIS*, pages 57–66, 2004.
- [21] M. Hagedoorn and R. C. Veldkamp. Reliable and efficient pattern matching using an affine invariant metric. *International Journal of Computer Vision*, 31:203–225, 1999.

- [22] D.P. Huttenlocher, G.A. Klauderman, and W.J. Rucklidge. Comparing images using the Hausdorff distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15:850–863, 1998.
- [23] B. Kuijpers and S. Haesevoets. Closure properties of classes of spatio-temporal objects under boolean set operations. [1], pages 79–86.
- [24] B. Kuijpers, J. Paredaens, and D. Van Gucht. Towards a theory of movie database queries. In *Proceedings of the 7th International Workshop on Temporal Representation and Reasoning* [1], pages 95–102.
- [25] G. Nielson. A characterization of an affine invariant triangulation. In G. Farin, H. Hagen, and H. Noltemeier, editors, *Geometric Modelling, Computing Supplementum 8*, pages 191–210, 1993.
- [26] E. Novak and K. Ritter. Some complexity results for zero finding for univariate functions. *Journal of Complexity*, 9:15–40, 1993.
- [27] J. Paredaens, J. Van den Bussche, and D. Van Gucht. Towards a theory of spatial database queries. In *Proceedings of the 13th ACM Symposium on Principles of Database Systems*, pages 279–288. ACM Press, 1994.
- [28] J. Paredaens, G. Kuper, and L. Libkin, editors. *Constraint databases*. Springer-Verlag, 2000.
- [29] D. Pfoser and N. Tryfona. Requirements, definitions and notations for spatiotemporal application environments. In R. Laurini, K. Makki, and N. Pissinou, editors, *Proceedings of the 6th International Symposium on Advances in Geographic Information Systems (ACM-GIS'98)*, pages 124–130, 1998.
- [30] P. Revesz. *Introduction to Constraint Databases*. Springer-Verlag, 2002.
- [31] L.G. Roberts. Machine perception of three-dimensional solids. *J.T. Tippett, editor, Optical and Electro-optical Information Processing*, 1965.
- [32] J.T. Schwartz Y. Lamdan and H.J. Wolfson. Affine-invariant model-based object recognition. *IEEE Journal of Robotics and Automation*, 6:578–589, 1990.