2017

Affine-Invariant Triangulation of Spatio-Temporal Data with an Application to Image Retrieval

Sofie Haesevoets
sofie.haesevoets@luciad.be

Bart Kuijpers
UHasselt–Hasselt University and transnational University Limburg, bart.kuijpers@uhasselt.be

Peter Z. Revesz
University of Nebraska-Lincoln, prevesz1@unl.edu

International Journal of Geo-Information

MDPI

Article

Affine-Invariant Triangulation of Spatio-Temporal Data with an Application to Image Retrieval

Sofie Haesevoets [1], Bart Kuijpers [2,*] and Peter Z. Revesz [3]

1 Luciad, Gaston Geenslaan 11, 3001 Leuven, Belgium; sofie.haesevoets@luciad.be
2 Databases and Theoretical Computer Science Research Group, UHasselt—Hasselt University and Transnational University Limburg, Agoralaan, Gebouw D, 3590 Diepenbeek, Belgium
3 Department of Computer Science & Engineering, University of Nebraska-Lincoln, 256 Avery Hall, 1144 T Street, Lincoln, NE 68588-0115, USA; revesz@cse.unl.edu
* Correspondence: bart.kuijpers@uhasselt.be; Tel.: +32-(0)476-74-19-39

Academic Editors: Ozgun Akcay and Wolfgang Kainz
Received: 10 February 2017; Accepted: 25 March 2017; Published: 28 March 2017

Abstract: In the *geometric data model* for spatio-temporal data, introduced by Chomicki and Revesz , 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. The application of our affine-invariant spatial triangulation method to image indexing and retrieval is discussed and an experimental evaluation is given in the context of bird images.

Keywords: spatio-temporal data models; affine transformations; triangulations

1. Introduction and Summary

1.1. Introduction

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 [1–9]. Several data models for representing spatio-temporal data have been proposed already. In this article, we adopt the *geometric data model* that was introduced by Chomicki and Revesz [1] and of which closure properties under boolean operations were later studied by them and the present authors [4,10]. 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 what 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.

1.2. Summary of Results

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., [11]).

Our spatio-temporal triangulation is also *invariant under time-dependent affinities*. In the area of spatial database research, much attention has been payed to affine invariance of both data description and manipulation techniques and queries [12–14]. 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* [15] is widely adopted. This assumption says that when an object is repeatedly photographed from different viewpoints, and the object is relatively far away form 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 [16–18]. Also, in computer graphics, affine-invariant norms and triangulations have been studied [19]. In the field of spatial and spatio-temporal constraint databases [20,21], affine-invariant query languages [12–14] have been proposed. For spatial data, there exist several triangulation algorithms, but, apart from the triangulation of Nielson [19], 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.

Affine-invariant measures have also been applied to shape matching, shape retrieval, and symmetry detection problems [22]; to point matching in aerial imaging [23]; to visual features detection in computer vision [24]; and object recognition [25].

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.

Table 1. Summary of the complexity results.

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

We remark that such triangulations could also be computed via general purpose cell decomposition algorithms, most notably cylindrical algebraic decomposition [26]. 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. We end this introduction with a description of a number of application scenarios.

1.3. Application Scenarios

There is a range of possible applications for our proposed algorithms. Below we describe briefly some selected applications from ecology and criminology.

Application Scenario 1: Consider an ornithologist who is exploring a jungle and takes some photos of some new bird species. Suppose that the ornithologist would like to find in an image database of already known birds those bird images that are similar to the image of the new bird species. This content-based image retrieval problem is challenging because the images of the birds from even the same species may vary slightly, especially in size and due to changing view angles, for example slightly higher or lower. Hence, the image retrieval algorithm needs to be affine-invariant. In this paper, we provide the basis for an image retrieval algorithm that retrieves for the ornithologist all the images of birds that are similar to the image of the new bird species using a similarity measure that is able to disregard simple scaling and view angle distortions [27].

Application Scenario 2: Consider again the ornithologist. Sometimes closely related bird species can be best distinguished by their behavior, for example the way of they crack and eat some nut or by their mating dance. A famous example of this are the Galapagos finches that Charles Darwin has recognized to be several separate species with only slight changes in appearance (primarily focused on the shape and the size of their beaks) but with a very marked change in their behavior (the type of fruits and nuts they eat). Suppose that the ornithologist takes a short video clip of the new bird species eating some fruit and is interested in looking up in a video database all the known birds which are similar both in appearance and in their eating behavior. This is an even more challenging problem that requires the temporal sequence of the images (the video) to match as well while still allowing for the scaling and the view angle distortions that we mentioned in the previous application scenario.

Application Scenario 3: Videocameras are frequently used for crime control. Hence the appearance and way of walking of a criminal may be captured on a video and compared to the appearance and motion of already recorded persons in a video databases.

Application Scenario 4: In spite of all the security cameras, fingerprint analysis remains a major tool in criminal investigations because while criminals often disguise their appearance, they inadvertently leave their fingerprints on the objects that they touch. Usually each fingerprint can be broken into distinctive regions of swirls and arches. An arch could be approximated by a single triangle and a swirl can be decomposed into a hexagon and further into six equilateral triangles. This gives

a natural triangulation of fingerprint images. Such a triangulation forms a basis of our image similarity algorithms as opposed to other algorithms. The advantage of a triangulation is that each triangle can be further analyzed in more detail. For example, in each triangle of a fingerprint the number and thickness of the lines that define the tiny groves of the skin is very unique. Hence two fingerprints that roughly correspond in the location of swirls and arches can be distinguished by the patterns within each triangle.

Further applications are discussed in Section 5. Application scenarios 1 and 2 are elaborated in Section 6, where we present an application of our affine-invariant spatial triangulation method to image indexing and retrieval in the context of bird images.

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. The application of our affine-invariant spatial triangulation method to image indexing and retrieval is described in Section 6. We end with some concluding remarks in Section 7.

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 [1], 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 $\{(x_1, x_2, \ldots, x_d) \in \mathbb{R}^d \mid p(x_1, x_2, \ldots, x_d) > 0\}$, where p is a polynomial with integer coefficients in the real variables x_1, x_2, ..., x_d. Properties of semi-algebraic sets are well known [28].

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* $(\mathbb{R}^2 \times \mathbb{R})$. □

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 [28].

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.*

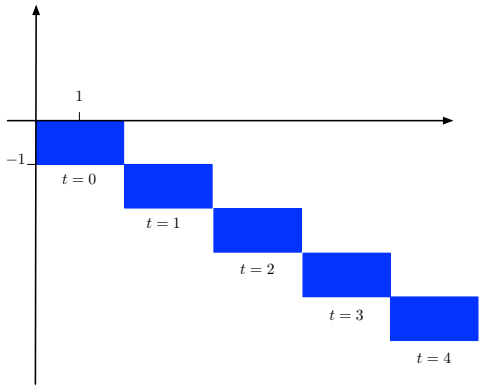

Figure 1. An example of a spatio-temporal object.

In *the geometric data model* [1,4], 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 lifespan; 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 [1,4]. In this article, we consider *spatio-temporal objects that can be represented as finite unions of triangles moved by time-dependent affine transformations*. We refer also to Definition 2. Other combinations have been studied [4] 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 [4]. 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 coefficients. For technical reasons, we allow a triangle to degenerate into a line segment or a point;*
- *$I \subset \mathbb{R}$ is the* time domain *(a point or an interval) of $\mathcal{O}$; and*
- *$f : \mathbb{R}^2 \times \mathbb{R} \to \mathbb{R}^2$ 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. □

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* $\{\mathcal{O}_1, \ldots, \mathcal{O}_n\}$ *of atomic geometric objects. It represents the spatio-temporal object* $\cup_{i=1}^{n} st(\mathcal{O}_i)$. □

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 $\{\mathcal{O}_1, \ldots, \mathcal{O}_n\}$ 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 $\overline{\bigcup}_{i=1}^{n} I_i$).

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* $(x,y;t) \mapsto (x+2t+2, y-t-1)$.

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.*

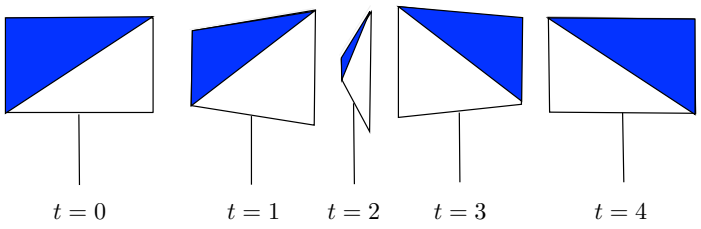

Figure 2. A spatio-temporal object (a traffic sign) shown at six moments.

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

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 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. We remark that, in the following definition, we consider filled triangles and we allow a triangle to degenerate into a closed line segment or a point.

We use the notion of the *interior* of a set 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.

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

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

We remark that in the second part of Definition 5, at each moment τ_0 in the time domain of $\{\mathcal{O}_1, \ldots, \mathcal{O}_n\}$, $\mathcal{T}_i^{\tau_0}$ 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 [11]).

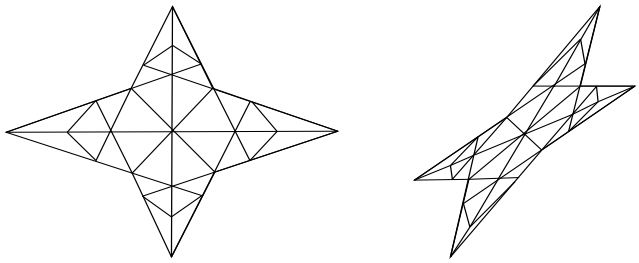

Figure 3. The triangulations of a snapshot (**left**) and of an affine transformation of the snapshot (**right**).

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 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 $\alpha : \mathbb{R}^2 \to \mathbb{R}^2$ 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 $\{\mathcal{O}_1, \ldots, \mathcal{O}_n\}$ and $\{\mathcal{O}'_1, \ldots, \mathcal{O}'_m\}$ for which for each moment τ_0 in their time domains, there is an affinity $\alpha_{\tau_0} : \mathbb{R}^2 \to \mathbb{R}^2$ such that if $\alpha_{\tau_0}(\{\mathcal{O}_1, \ldots, \mathcal{O}_n\}^{\tau_0}) = \{\mathcal{O}'_1, \ldots, \mathcal{O}'_m\}^{\tau_0}$, also $\alpha_{\tau_0}(\mathcal{T}_{ST}(\{\mathcal{O}_1, \ldots, \mathcal{O}_n\})^{\tau_0}) = \mathcal{T}_{ST}(\{\mathcal{O}'_1, \ldots, \mathcal{O}'_m\})^{\tau_0}$. □

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.

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.

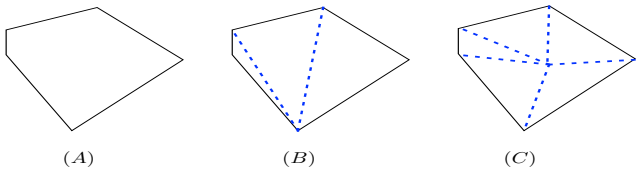

Figure 4. Two different triangulations (in (**B**,**C**)) of a convex polygon (in (**A**)).

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 5A, 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. Therefor, 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 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 5D, 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 5D 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.

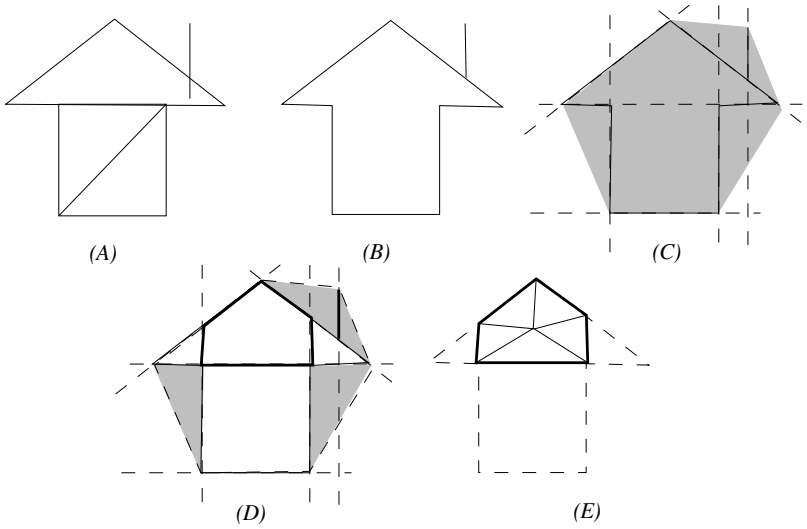

Figure 5. An illustration of the different steps in the spatial triangulation algorithm. The subfigures (**A**–**E**) are explained in the text.

In the detailed description of the algorithm, we will use some well known techniques. One of those is the *doubly-connected edge list* [29], 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.* □

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 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 **a** *stores the coordinates of* **a** *and a pointer to an arbitrary half-edge that has* **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 $e \rightarrow$ origin of e are $e \rightarrow$ origin $\rightarrow x$ and $e \rightarrow$ origin $\rightarrow y$.

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

Out:= $\emptyset$.
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.
Compute the convex hull $\mathcal{CH}(S)$ of S.
Construct the doubly connected edge list DCEL(S), induced by the planar subdivision defined by the lines through the segments of $\mathcal{B}(S)$, using $\mathcal{CH}(S)$ as a bounding box.
while there are any unvisited half-edges in DCEL($\mathcal{S}$) left **do**
  Let e be an unvisited edge.
  $\Sigma_x := 0$, $\Sigma_y := 0$, count := 0, Elist := $\emptyset$.
  while e is unvisited **do**
    Mark e with the label *visited*.
    Elist := Elist $\cup$ $\{(e \rightarrow \text{origin}, e \rightarrow \text{next} \rightarrow \text{origin})\}$, $\Sigma_x := \Sigma_x + e \rightarrow \text{origin} \rightarrow x$, $\Sigma_y := \Sigma_y + e \rightarrow \text{origin} \rightarrow y$, count := count + 1.
    $e := e \rightarrow$ next.
  end while
  $\mathbf{x} := (\frac{\Sigma_x}{\text{count}}, \frac{\Sigma_y}{\text{count}})$.
  if the point **a** in **x** belongs to a face of $\mathcal{U}(S)$ **then**
    for all elements $(\mathbf{a}_s, \mathbf{a}_e)$ of Elist **do**
      Out := **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 **a**, $\mathbf{a}_s$ and $\mathbf{a}_e$.
    end for
  end if
end while
return Out.

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.

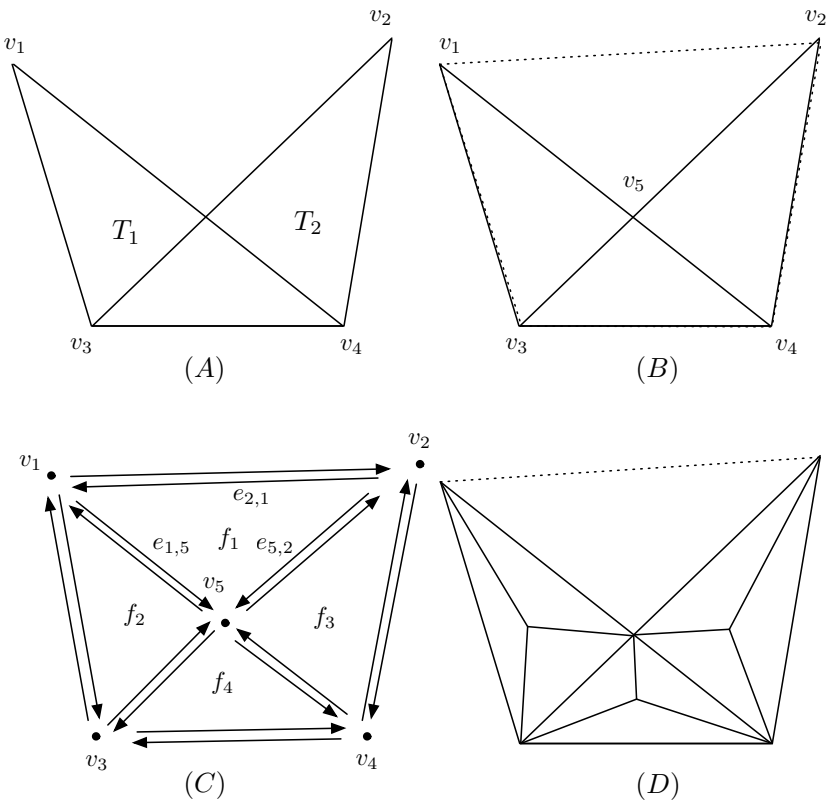

Figure 6. The different steps of Algorithm 1 applied to $S = \{T_1, T_2\}$. Figure (**A**) represents the input data. In this example, all boundary segments of all triangles of S contribute to the boundary of S. The line arrangement induced by the carriers of the edges of the input triangles is bounded 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**).

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

Half-Edge	Origin	Twin	Incident Face	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	$e_{4,2}$	$e_{2,5}$

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) 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.

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 $\{T_1, T_2, \ldots, T_k\}$ determine a snapshot S. It is easy to see that the output $\mathcal{T}_S(S) = \{T'_1, T'_2, \ldots, T'_\ell\}$ 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. □

Property 2 (Quadratic output complexity). *Let a snapshot S be given by the set $\{T_1, T_2, \ldots, T_m\}$, 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., [30]), 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 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 DCEL($\mathcal{S}$) (connecting that half-edge with the center of mass of the face it bounds). Therefor, the output contains $O(m^2)$ triangles. □

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 $\{T_1, T_2, \ldots, T_m\}$, 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 $\{T_1, T_2, \ldots, T_m\}$. Using a plane-sweep algorithm [31], 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 [32]). The same authors propose, in [30], 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 [30] 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 [32], already outputs the corner points of the convex hull in circular order. In the original algorithm [30] 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 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 [33]. 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 $O(m^2 \log m)$. □

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

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

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 DCEL($\mathcal{S}$)	$O(m^2)$
5–19	Polygon extraction and triangulation	$O(m^2 \log m)$
	Overall time complexity	$O(m^2 \log m)$

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 $\{T_{a,1}, T_{a,2}, \ldots, T_{a,k}\}$ and B be a snapshot given by the set of triangles $\{T_{b,1}, T_{b,2}, \ldots, T_{b,\ell}\}$, such that there exists an affinity $\alpha : \mathbb{R}^2 \to \mathbb{R}^2$ 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 $\mathcal{T}_S(B)$.

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. □

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 [34]. 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 $h : \mathbb{R}^2 \to \mathbb{R}^2$ 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 $h(T_{i,((i+1) \bmod 3)})$.* □

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.*

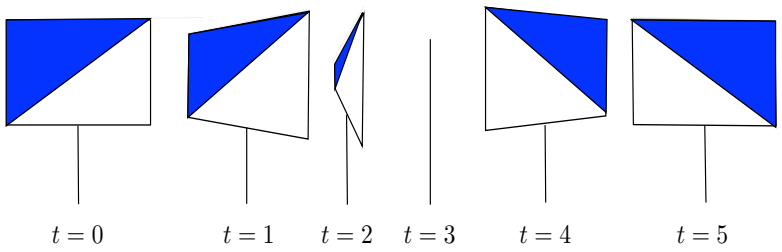

Figure 7. Snapshots of a traffic sign as seen by an observer circularly moving around it.

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

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).*

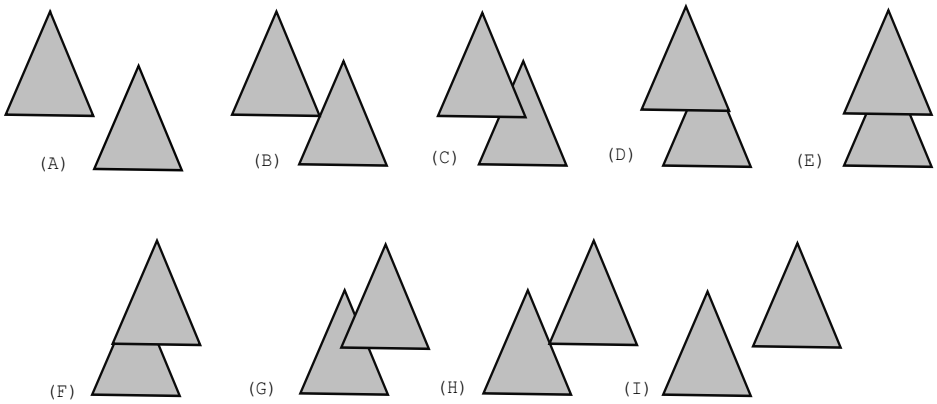

Figure 8. The snapshots at time moments $t = \frac{1}{4}$ (**A**), $t = \frac{1}{2}$ (**B**), $t = 1$ (**C**), $t = \frac{3}{2}$ (**D**), $t = 2$ (**E**), $t = \frac{5}{2}$ (**F**), $t = 3$ (**G**), $t = \frac{7}{2}$ (**H**) and $t = 4$ (**I**) of the geometric object of Example 8.

Let $\mathcal{O} = \{\mathcal{O}_1 = (S_1, I_1, f_1), \mathcal{O}_2 = (S_2, I_2, f_2), \ldots, \mathcal{O}_m = (S_m, I_m, f_m)\}$ 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 $i = 1, \ldots, m$.

4.1. The Partitioning Step

Let $\mathcal{O} = \{\mathcal{O}_1 = (S_1, I_1, f_1), \mathcal{O}_2 = (S_2, I_2, f_2), \ldots, \mathcal{O}_m = (S_m, I_m, f_m)\}$ be a geometric object. In the first step of $\mathcal{T}_{ST}$, the time domain I of $\mathcal{O}$, i.e., the convex closure $\bigcup_{i=1}^{m} I_i$ of the union of all the time

domains $I_i (i = 1, \ldots, m)$ 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 $car(f_i(S_i, \tau))$, $(i = 1, \ldots, m)$.

In [4], 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* $\mathcal{O} = \{\mathcal{O}_1, \mathcal{O}_2, \ldots, \mathcal{O}_m\}$ *any partition of the interval* $I = \bigcup_{i=1}^{m} I_i$ *into a finite number of time intervals* $J_1, \ldots, J_k$ *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* $\mathbb{R}^2$. □

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 [4].

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

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 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. Therefor, 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*

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

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

1: Let $\chi = (\tau_1 \leq \tau_2 \leq \ldots \leq \tau_k)$ (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.

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.

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

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 $\mathcal{O} = \{\mathcal{O}_1 = (S_1, I_1, f_1), \mathcal{O}_2 = (S_2, I_2, f_2), \ldots, \mathcal{O}_n = (S_n, I_n, f_n)\}$ 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 $I = \overline{\bigcup}_{i=1}^{n} I_i$ containing $O(n^3 d)$ 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, $O(n^2 d)$ 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, $O(n^3 d)$ elements are added to χ.

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

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 $O(d^2 \log d \log\log(\frac{1}{\epsilon}))$ [35]. 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 $\mathcal{O} = \{\mathcal{O}_1 = (S_1, I_1, f_1), \mathcal{O}_2 = (S_2, I_2, f_2), \ldots, \mathcal{O}_n = (S_n, I_n, f_n)\}$ 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 $I = \bigcup_{i=1}^{n} I_i$ in time $O(n^3(z(d, \epsilon) + d \log n))$.*

Proof. Let $\mathcal{O} = \{\mathcal{O}_1 = (S_1, I_1, f_1), \mathcal{O}_2 = (S_2, I_2, f_2), \ldots, \mathcal{O}_n = (S_n, I_n, f_n)\}$ 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 $z(d, \epsilon)$. Therefor, 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 $O(n^3(z(d, \epsilon) + d \log n))$. □

We now proceed with the triangulation step.

4.2. The Triangulation Step

Starting with a geometric object $\mathcal{O} = \{\mathcal{O}_1 = (S_1, I_1, f_1), \mathcal{O}_2 = (S_2, I_2, f_2), \ldots, \mathcal{O}_n = (S_n, I_n, f_n)\}$, the partitioning algorithm identifies a list χ of time moments that is used to partition the time domain

$I = \bigcup_{i=1}^{n} I_i$ 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 $\mathcal{O}^\tau$. 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 $]\tau_j, \tau_{(j+1)}[$ defined by two subsequent elements 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 $\mathcal{O}^{\tau_m}$ 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 $\mathcal{O}^{\tau_m}$ 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 $\mathcal{O}^{\tau_m}$ 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 [4]).

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.

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

for all time moments $\tau_j, j = 1 \ldots k$, of χ **do**
  for all triangles T in $\mathcal{T}_S(\mathcal{O}^{\tau_j})$ **do**
    return the atomic element $(T, \{\tau_j\}, Id)$.
  end for
end for
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.
Let S_{Active} be a list of elements of the type *Segments*, $S_{\text{Active}} = ()$.
for all pairs $(\tau_j, \tau_{j+1}), j = 1 \ldots (k-1))$, in χ **do**
  $\tau_m := \frac{1}{2}(\tau_j + \tau_{j+1})$.
  Remove all elements (Pt_1, Pt_2) of S_{Active} for which $\tau_j = Pt_1 \to I_e = Pt_2 \to I_e$.
  for all elements (Pt_1, Pt_2) remaining in S_{Active} **do**
    $Pt_r \to Point := (Pt_r \to f_x(\tau_m), Pt_r \to f_y(\tau_m)), r = 1, 2.$
  end for
  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 τ_i **do**
    Construct three *Points* Pt_1, Pt_2 and Pt_3 such that $Pt_r \to Point = \mathbf{a}_r$, $Pt_r \to f_x = f_i(a_{r,x}, \tau_m)$, $Pt_r \to f_y = f_i(a_{r,y}, \tau_m)$ and $Pt_r \to I_b$ and $Pt_r \to I_e$ respectively contain τ_j and τ_{j+1} $(r = 1, \ldots, 3)$.
    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_{Active}.
  end for
  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_{Active}. Meanwhile, construct the subdivision $\mathcal{U}(\mathcal{O}^{\tau_m})$.
  Compute the convex hull $\mathcal{CH}^t(S_{\text{Active}})$, using only the constant point information of the elements of S_{Active}, a list of elements of the type *Points*.
  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 intersection point.
  while there are any unvisited *Segments* St in $\text{DCEL}^t(\mathcal{S})$ left **do**
    Compute the list E^tlist 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
    if $Pt_m \to Point$ belongs to a face of $\mathcal{U}(\mathcal{O}^{\tau_m})$ **then**
      for all elements $St = (Pt_1, Pt_2)$ of E^tlist **do**
        Output the atomic object (S, I, f), where S is the triangle with corner points $Pt_1 \to Point$, $Pt_2 \to Point$ and $Pt_m \to Point$ and I is $]\tau_j, \tau_{j+1}[$. The transformation function f is computed using the functions $Pt_1 \to f_x$, $Pt_1 \to f_y$, $Pt_2 \to f_x$, $Pt_2 \to f_y$, $Pt_m \to f_x$ and $Pt_m \to f_y$.
      end for
    end if
  end while
end for

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* $(x+t,y)$.

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 [4]. 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* $(x - \frac{1}{4} + t, y)$.

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

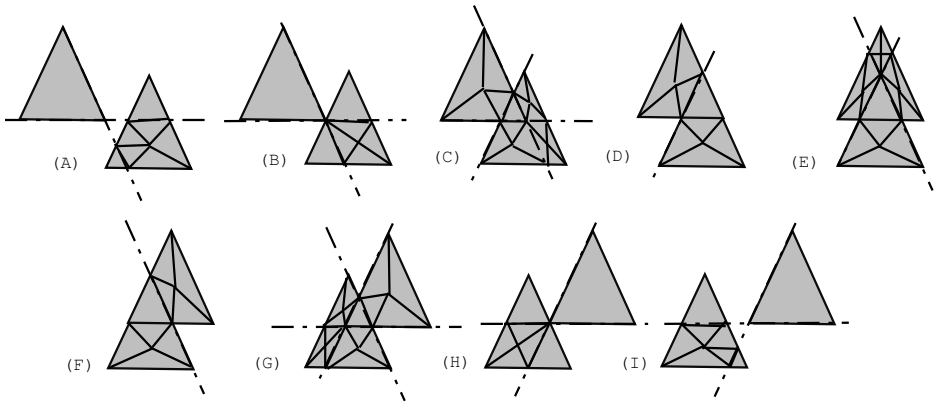

Figure 9. The triangulations of the objects of Example 8 at time moments $t = \frac{1}{4}$ (**A**), $t = \frac{1}{2}$ (**B**), $t = 1$ (**C**), $t = \frac{3}{2}$ (**D**), $t = 2$ (**E**), $t = \frac{5}{2}$ (**F**), $t = 3$ (**G**), $t = \frac{7}{2}$ (**H**) and $t = 4$ (**I**).

Property 8 (Triangulation step: output complexity). *Given a geometric object* $\mathcal{O} = \{\mathcal{O}_1 = (S_1, I_1, f_1), \mathcal{O}_2 = (S_2, I_2, f_2), \ldots, \mathcal{O}_n = (S_n, I_n, f_n)\}$ *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 their are $O(k)$ moments and intervals for which we have to consider such a triangulation, or a slightly adapted version of it, this gives $O(n^2k)$. □

Property 9 (Triangulation step: computational complexity). *Let a geometric object* $\mathcal{O} = \{\mathcal{O}_1 = (S_1, I_1, f_1), \mathcal{O}_2 = (S_2, I_2, f_2), \ldots, \mathcal{O}_n = (S_n, I_n, f_n)\}$ *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 $O(kz(d,\epsilon)n^2 \log n)$. □

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 $\lim_{t \to \tau} f_1(S_1, t) = S_2$. We note that, for rational functions f of t, $\lim_{t \to \tau} f(t)$ equals $f(\tau)$, provided that τ is in the domain of f. We also note 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 $t = \tau$.

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 ([4]).

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.

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

Sort all atomic objects $\mathcal{O}_i$ by their time domains.
Let χ' be the list $(\tau_1,]\tau_1, \tau_2[, \tau_2, \ldots,]\tau_{k-1}, \tau_k[, \tau_k)$.
Let J_1 be the first element of χ' and J_2 the second.
while there are any elements in χ' left **do**
  $\mathcal{S}_1$ (resp. $\mathcal{S}_2$) is the set of all objects having J_1 (resp. J_2) as their time domain.
  if J_1 is a point **then**
    Preprocess the reference objects of the elements of $\mathcal{S}_1$ such that we can search the planar subdivision $\mathcal{U}_1$ they define.
    let **Found** be true.
    for all objects $\mathcal{O}_i = (S_i, J_2, f_i)$ in $\mathcal{S}_2$ **do**
      Check whether J_1 is part of the time domain of f_i.
      Compute their snapshot at time J_1 (which is a triangle T).
      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.
      if **Found** is false **then**
        break;
      end if
    end for
    if **found** is true **then**
      remove all elements of $\mathcal{S}_1$ from $\mathcal{O}$ and extend the time domain of all elements of $\mathcal{S}_2$ to $J_1 \cup J_2$.
      $J_1 = J_1 \cup J_2$ and J_2 is the next element of χ' if any exists.
    else
      $J_1 = J_2$ and J_2 is the next element of χ', if any exists.
    end if
  else
    if J_2 is a point **then**
      do the same as in the previous case, but switch the roles of J_1 and J_2.
    else
      Let J'_1 be the element of $\{J_1, J_2\}$ the form $(\tau'', \tau[$ and J'_2 the one of the form $[\tau, \tau')$.
      Check whether $(\tau'', \tau[$ and $\{\tau\}$ can be merged.
      if this can be done **then**
        Check for each pair of matching atomic objects whether their transformation functions are the same ([4]).
      end if
    end if
  end if
end while

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* $(x+t,y)$.

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.*

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* $\mathcal{O} = \{\mathcal{O}_1 = (S_1, I_1, f_1), \mathcal{O}_2 = (S_2, I_2, f_2), \ldots, \mathcal{O}_n = (S_n, I_n, f_n)\}$, *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 [33]. 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 [33]. 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 $O(\frac{n}{K} \log \frac{n}{K} + \frac{n}{K} z(d,\epsilon))$.

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 $O(n \log \frac{n}{K} + nz(d,\epsilon))$. □

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

The following property follows from Property 6 and Property 8.

Property 11 ($\mathcal{T}_{ST}$: output complexity). *Given a geometric object* $\mathcal{O} = \{\mathcal{O}_1 = (S_1, I_1, f_1), \mathcal{O}_2 = (S_2, I_2, f_2), \ldots, \mathcal{O}_n = (S_n, I_n, f_n)\}$ *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* $O(n^5 d)$ *atomic objects.*

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

χ = **Partition**(O);
$\{\mathcal{O}''_1, \mathcal{O}''_2, \ldots, \mathcal{O}''_m\}$ = **Triangulate**($\mathcal{O}$, χ);
if χ has more than one element **then**
    $\{\mathcal{O}'_1, \mathcal{O}'_2, \ldots, \mathcal{O}'_\ell\}$ = **Merge**($\{\mathcal{O}''_1, \mathcal{O}''_2, \ldots, \mathcal{O}''_m\}, \chi$);
    return $\{\mathcal{O}'_1, \mathcal{O}'_2, \ldots, \mathcal{O}'_\ell\}$.
else
    return $\{\mathcal{O}''_1, \mathcal{O}''_2, \ldots, \mathcal{O}''_m\}$.
end if

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

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.

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

Property 12 ($\mathcal{T}_{ST}$**: computational complexity).** *Given a geometric object* $\mathcal{O} = \{\mathcal{O}_1 = (S_1, I_1, f_1), \mathcal{O}_2 = (S_2, I_2, f_2), \ldots, \mathcal{O}_n = (S_n, I_n, f_n)\}$ *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 $\mathcal{O} = \{\mathcal{O}_1, \ldots, \mathcal{O}_n\}$ and $\mathcal{O}' = \{\mathcal{O}'_1, \ldots, \mathcal{O}'_m\}$ be geometric objects for which for each moment τ_0 in their time domains, there is an affinity $\alpha_{\tau_0} : \mathbb{R}^2 \to \mathbb{R}^2$ such that $\alpha_{\tau_0}(\{\mathcal{O}_1, \ldots, \mathcal{O}_n\}^{\tau_0}) = \{\mathcal{O}'_1, \ldots, \mathcal{O}'_m\}^{\tau_0}$.

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

The following corollary follows straightforwardly from Property 13:

Corollary 1. *Let $\mathcal{O} = \{\mathcal{O}_1, \ldots, \mathcal{O}_n\}$ and $\mathcal{O}' = \{\mathcal{O}'_1, \ldots, \mathcal{O}'_m\}$ be two geometric objects such that there is an affinity $\alpha : \mathbb{R}^2 \to \mathbb{R}^2$ 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 $\mathcal{T}_{ST}(\mathcal{O}')$.* □

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.* □

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 $\{B\}$.

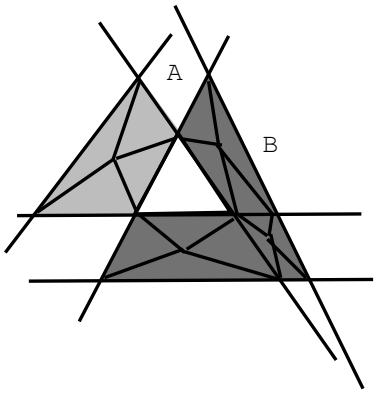

Figure 10. The set operations between objects are pre-computed in the triangulation.

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. An Application of Affine-invariant Triangulation to Image Retrieval

In this section, we describe an application of the affine-invariant spatial triangulation algorithm of Section 3 to the problem of image indexing and retrieval. We start by describing some affine-invariant color feature extraction methods in Section 6.1. Next, in Section 6.2, we describe the implementation and application for image retrieval in the context of bird images. Finally, in Section 6.3, we discuss an experimental evaluation of the proposed method.

6.1. Affine-Invariant Color Feature Extraction

We start with the description of two methods for color feature extraction. The *primary color ratio measure* computes the ratios of the sum of the red, respectively green and blue color values of all pixels, whereas the *rainbow color ratio measure* defines N colors a priori and computes the ratios of the areas of the picture filled with each color. We show that both measures are affine-invariant. We assume that each image is represented as a set of colored points. Most pixel-based image representations allow 256 different shades of green, red and blue.

6.1.1. The Primary Color Ratio Similarity Measure

Let R_A (respectively G_A, B_A) be the sum of all red (respectively green, blue) color values of the pixels of an image A. Then the ratios $\frac{R_A}{G_A}$ and $\frac{B_A}{G_A}$ (if $G_A \neq 0$) are independent of each other. Hence, we can describe each image A as a two-dimensional vector $V_A = (\frac{R_A}{G_A}, \frac{B_A}{G_A})$, which we call the *primary color ratio vector*. It is very unlikely that all three values R_A, G_A and B_A are zero for some image A. This would mean that the image is completely black. Therefore, we can always find one of R_A, G_A and B_A that is not zero and use this value in the denominator for computing the ratios. We assume that each image has some green in it.

We now prove that the primary color ratios are affine-invariant.

Property 14. *Let R_A, G_A and B_A be the sums of the red, green and blue pixel values of image A. Let α be an affinity. Then $R_{\alpha(A)} = |\alpha|R_A$, $G_{\alpha(A)} = |\alpha|G_A$ and $B_{\alpha(A)} = |\alpha|B_A$, where $|\alpha|$ is the determinant of the transformation matrix of α.* □

Proof. Let R_A be the sum of the red values of the pixels of image A. Then we can write R_A as $\sum_p r(p)$, where p ranges over all pixels of A and $r(p)$ is the red value of pixel p.

Let $|A|$ be the number of pixels in image A. There are 256 different possible values of red, so the amount of different values of red in image A is at most $min(|A|, 256)$. For each value of red r_i in image A, we can compute the number of pixels N_{r_i} occupied by that value. We have that $R_A = \sum_p r(p) = \sum_i r_i N_{r_i}$, where i ranges over all different values of red in the image.

If we apply transformation α to image A, we know from Chapter 21 of [21] that for each area composed of pixels with red value r_i, the area occupied by the pixels with red value r_i in $\alpha(A)$ equals $|\alpha|N_{r_i}$, where $|\alpha|$ is the determinant of the transformation matrix of α.

This means that:

$$R_{\alpha(A)} = \sum_i r_i |\alpha| N_{r_i} = |\alpha| \sum_i r_i N_{r_i} = |\alpha| \sum_p r(p) = |\alpha| R_A.$$

The same reasoning applies for the sum of the green, respectively blue, color values. □

Property 15. *Let R_A, G_A and B_A the sums of the red, green and blue pixel values of image A. Let α be an affinity. Then all ratios of two of these values are affine-invariant.* □

Proof. We only prove the Property for the ratios $\frac{R_A}{G_A}$ and $\frac{B_A}{G_A}$. More specific this means we have to prove that

$$\frac{R_A}{G_A} = \frac{R_{\alpha(A)}}{G_{\alpha(A)}} \text{ and } \frac{B_A}{G_A} = \frac{B_{\alpha(A)}}{G_{\alpha(A)}}.$$

We know from Property 14 that $R_{\alpha(A)} = |\alpha|R_A$, $G_{\alpha(A)} = |\alpha|G_A$ and $B_{\alpha(A)} = |\alpha|B_A$, where $|\alpha|$ is the determinant of the transformation matrix of α. By taking the ratios we get the required result. The proofs for all other ratios are similar. □

Definition 12. *Let V_A and V_B be the primary color ratio vectors of images A and B, respectively. Then the* primary color ratio similarity measure *defines the distance between A and B as the distance between the two-dimensional vectors V_A and V_B.* □

Remark that any distance measure can be used. In our implementation we use the Euclidean distance between the primary color ratio vectors.

6.1.2. The Rainbow Color Ratio Similarity Measure

The rainbow color ratio similarity measure was first introduced in Chapter 21 of [21]. From all possible red, green and blue values of the image pixels, a large set of colors of the rainbow can be defined. Suppose we classify each part of an image as having one of N different colors. The ratio of the total areas of any pair of colors is affine-invariant [21].

Suppose that we have a set of N different colors. We can derive $M = N - 1$ ratios that are independent of each other, e.g. by fixing one color and use it as the denominator for all ratios. We can describe each image as a vector of M values where the i^{th} entry corresponds to the value of the i^{th} rainbow color ratio.

The *rainbow color ratio similarity measure* is defined as follows [21]:

Definition 13. *Let V_A and V_B be the vectors of the rainbow color ratios in images A and B, respectively. Then the distance between A and B is defined as the Euclidean distance between V_A and V_B.* □

The primary color ratio measure treats the red, green and blue color values independently, but is a very global method. The rainbow color ratio measure results in a higher-dimensional feature vector, hence the vectors are possibly more sparsely distributed. However, the definition of colors out of their red, green and blue values implies that they are not completely independent from each other. In Section 6.3.1 we compare both methods experimentally.

6.1.3. Indexing Feature Vectors

The primary color ratio measure and rainbow color ratio measures described above construct a feature vector from an image. There exist several efficient techniques for indexing multidimensional points. In general, spatial access methods (SAM) can be divided into two classes: data-driven and space-driven. Examples of data-driven techniques are $R-$trees [36], R^*-trees [37] and Hilbert-$R-$trees [38]. Those are all based on a hierarchy of minimum bounding rectangles (MBRs). Quad trees [39] are an example of a space driven indexing technique. Quad trees, in the case of two-dimensional points, recursively partition the plane into four equal parts, until each part only contains one element. Quad trees are generally used when the feature vectors are two-dimensional, but they can be extended for higher dimensions too.

6.2. Implementation and Application for Image Retrieval of Bird Images

In previous sections, we introduced an affine-invariant comparison method for both color and shape information of an image. We now propose a two-level indexing structure based on both methods. First, a color ratio measure is used to group all images with comparable color schemes (up to

some threshold). Within such a group of similar images, shape information is used for more refined comparison.

We now describe in more detail the implementation of the measures. We used Intel's openCV library for image processing functions. The test set consists of colored drawings of birds, all having a white background. The images are obtained from *http://www.clipart.com*. The test set consists of 292 different images, and 10 of them have 9 distortions. The distortions include three rotations, two scalings, two sheer transformations and two reflections. Figure 11 shows an image and its distortions.

Figure 11. An example of an image representing a cockatiel and its distortions.

6.2.1. Implementation of the Primary Color and Rainbow Color Ratio Measures

The implementation of the primary color ratio measure is straightforward. We used the ratios $\frac{R}{G}$ and $\frac{B}{G}$. For the rainbow color ratio measure, we defined 9 colors: red, green, blue, yellow, turquoise, purple, white, gray and black. Their respective areas are denoted R, G, B, Y, T, P, W, Gr and Bl respectively.

The definition in terms of primary color range for the nine colors we used for the rainbow color ratio measure are as follows:

Color	**Formula**
red	$(R-G) \geq 50 \wedge (R-B) \geq 50$
yellow	$R \cong_c G \wedge (R-B) \geq 50$
green	$(G-R) \geq 50 \wedge (G-B) \geq 50$
turquoise	$B \cong_c G \wedge (B-R) \geq 50$
blue	$(B-G) \geq 50 \wedge (B-R) \geq 50$
purple	$R \cong_c B \wedge (R-G) \geq 50$
white	$R \cong_c G \cong_c B \wedge max(R,G,B) \leq 30$
grey	$R \cong_c G \cong_c B \wedge \neg(max(R,G,B) \leq 30 \vee max(R,G,B) \geq 220)$
black	$R \cong_c G \cong_c B \wedge max(R,G,B) \geq 220.$

In the above, the expression $X \cong_c Y$ denotes that the absolute value of the difference between the values X and Y is at most 20.

For each image, we take out the background. Let I be the total area of the picture without its background. The rainbow color ratio vector of each image consists of the ratios $\frac{R}{I}$, $\frac{G}{I}$, $\frac{B}{I}$, $\frac{Y}{I}$, $\frac{T}{I}$, $\frac{P}{I}$, $\frac{Gr}{I}$ and $\frac{Bl}{I}$.

For both methods, we do not take into account the pixel values of the background of the image.

6.2.2. Implementation of the Shape Comparison Measure Based on the Affine Triangulation

As we use digital images, we don't have an exact description of the shape by means of boundary line segments. We extract the boundary information of an image and compute the triangulation using the *Affine Shape Comparison (ASC) Algorithm* (Algorithm 6).

In our experiments, we reduce the number of lines L to 10. Figure 12 shows a parrot and the lines which result using Algorithm 1 of Section 3.

Algorithm 6 The ASC Algorithm

Step 1. The image is converted into black and white to abstract from all color information and get the shape silhouette;

Step 2. The shape boundary is detected using a Canny edge detector operation. This method, proposed by Canny [40], is considered the most efficient method for edge detection. It is implemented in the openCV library;

Step 3. From this boundary, lines are extracted using the Simple Hough Transform (SHT) [41]. This is a popular method for extracting geometric primitives. Each line in an image corresponds to a point (ρ, θ) in the Hough space such that the equation of the line is $\rho = xcos(\theta) + ysin(\theta)$. Given a threshold τ, the Hough transform returns $(\rho, \theta)-$pairs corresponding to lines that contain at least τ boundary points.

Step 4. The lines are then clipped against the image frame and *merged*.
The nature of the Hough transform is such that it returns a lot of redundant lines. If the image contains a strong line, then it is detected as a bundle of lines that vary only slightly. Ideally, there would be only a limited amount of lines. Otherwise, the triangles resulting from the triangulation become very small and the chance that two different images match increases because of the small triangles. Therefore, we merge the lines until there are at most L left. The input of the merging procedure is a set of line segments. First, all lines receive weight one. The weight of line segment $\overline{\mathbf{p}, \mathbf{q}}$ is denoted $W(\mathbf{p}, \mathbf{q})$. A threshold is initialized on one pixel. Repeatedly this threshold is increased until no more than L lines are left. For each value of the threshold, all pairs of line segments $\overline{\mathbf{pq}}$ and $\overline{\mathbf{rs}}$ are tested whether their end points are closer together than the threshold value. If so, the pair of segments is replaced by one segment $\overline{\mathbf{uv}}$ that is computed as follows:

$$\mathbf{u} = \frac{\mathbf{W}(\mathbf{p}, \mathbf{q})}{\mathbf{W}(\mathbf{p}, \mathbf{q}) + \mathbf{W}(\mathbf{r}, \mathbf{s})}\mathbf{p} + \frac{\mathbf{W}(\mathbf{r}, \mathbf{s})}{\mathbf{W}(\mathbf{p}, \mathbf{q}) + \mathbf{W}(\mathbf{r}, \mathbf{s})}\mathbf{r}$$

and

$$\mathbf{v} = \frac{\mathbf{W}(\mathbf{p}, \mathbf{q})}{\mathbf{W}(\mathbf{p}, \mathbf{q}) + \mathbf{W}(\mathbf{r}, \mathbf{s})}\mathbf{q} + \frac{\mathbf{W}(\mathbf{r}, \mathbf{s})}{\mathbf{W}(\mathbf{p}, \mathbf{q}) + \mathbf{W}(\mathbf{r}, \mathbf{s})}\mathbf{s}.$$

Step 5. The four segments bounding the image frame are added and the triangulation is computed as described in Algorithm 1 of Section 3.

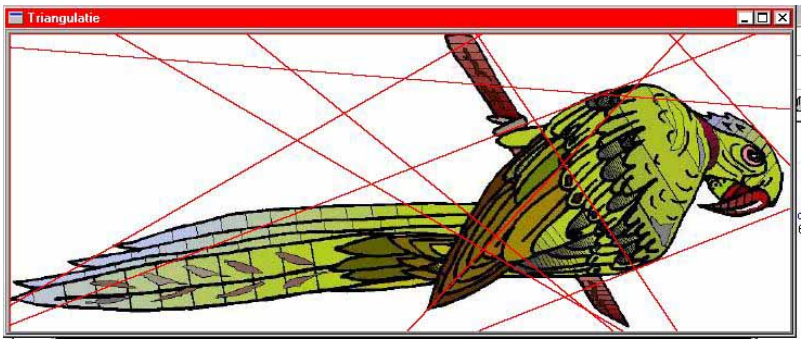

Figure 12. The detected lines after reduction.

Since we are dealing with very approximate boundary information, it is not robust to compare the exact topological information of the triangulations of the two images. Instead, we compute the color ratio vectors of the N biggest polygons (i.e., we don't compute the last step of the triangulation algorithm). As relative areas are affine-invariant, it is true that if $p_1, p_2, \ldots, p_N$ are the N biggest polygons in the triangulation of image A, and α is an affine transformation, then $\alpha(p_1), \alpha(p_2), \ldots, \alpha(p_N)$ are the biggest polygons of the triangulation of image $\alpha(A)$.

Finding a one-to-one mapping between the biggest polygons would have to deal with the case that several polygons have Reni, we take the weighted average of the N feature vectors for each image, and compare those. We call those vectors *weighted color ratio vectors*. The weight of each polygon is the ratio of the area of that polygon and the total area of the three biggest polygons (hence the weight is also an affine invariant). If N is large, then there could be a large difference in the size of the biggest and the smallest polygons. If N is small, then in general there is only a small difference and in that case the weight may be even eliminated without a significant loss of performance. The shape similarity measure is defined as the distance (e.g., the Euclidean distance) between two weighted color ratio vectors.

6.3. Experimental Results

We now describe the experiments and their results.

6.3.1. Comparison of the Color Ratio Methods

We compare the primary color ratio and the rainbow color ratio measure. We compare the distribution of the values of the distances between all pairs of images. Also, we test for each of the 10 images of which there are distortions in the database, how accurate all distortions can be found using each method.

We first compare the distribution of the values of the distances between all pairs of images. Figure 13 shows this distributions for each method. The horizontal axis shows the range of the distance values. The vertical axis shows the number of image pairs that have a distance within a certain range. It appears that the rainbow color ratio measure has a smaller range of distance values, they are all smaller than 1.3. For the primary color ratio measure, the values range between zero and 2.7 (for clarity, in the graph of Figure 13, all values greater than 2 are put in one category).

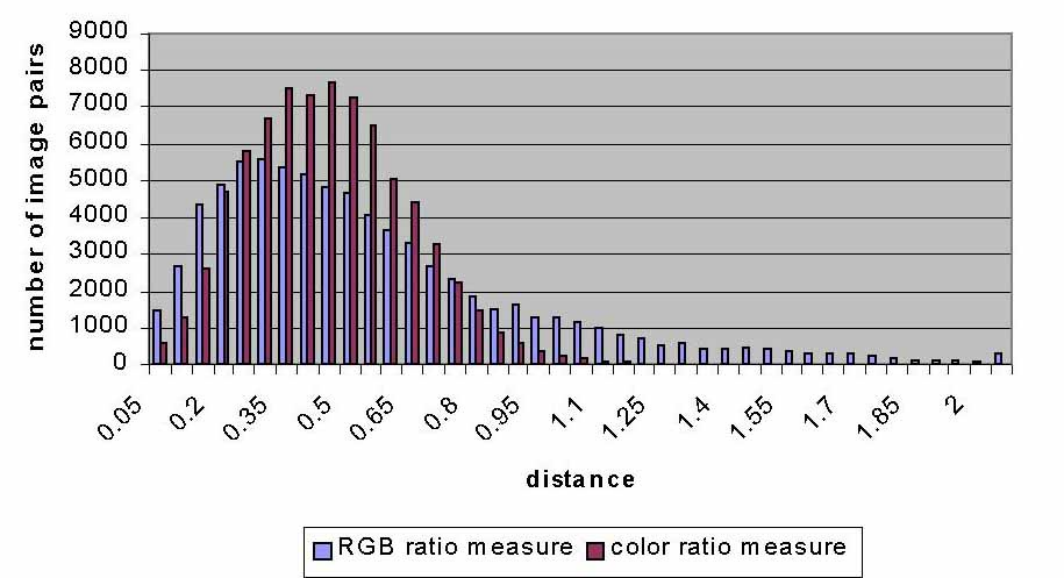

Figure 13. Comparison of both color measures: distribution of the distance values.

We now compute, for each of the 10 test images, the distances to all other images and sort the other images by that distance. We count the maximal distance between each image and its 9 distortions, and the number of images (called *rank*) we have to return in order to get all relevant matches, i.e., the 9 distortions. These calculations are shown in the table below, for both the primary color ratio and the rainbow color ratio. Figure 14 shows for both measures the closest images for a parrot, corresponding to image one in the table. The upper row is the result for the primary color ratio measure, the lower row is the rainbow color ratio measure. We see that the primary color ratio

measure has better performance. The average distance of the primary color ratio measure is only half that of the rainbow color ratio measure. If we compute the efficiency of both methods we have that the rainbow color ratio measure has an efficiency of 60%, where the efficiency of the primary color ratio is 89%. The following table gives an overview.

	Primary		**Rainbow**		**Combined**	
id	**rank**	**dist.**	**rank**	**dist.**	**rank**	**scan**
1	11	0.0486	14	0.2172	9	14
2	9	0.1793	10	0.9633	9	10
3	9	0.0293	11	0.0931	9	11
4	9	0.0096	21	0.1203	10	21
5	10	0.0274	9	0.0337	9	10
6	9	0.0619	10	0.0919	9	10
7	16	0.0212	31	0.1511	9	31
8	9	0.1078	23	0.1027	9	23
9	10	0.0413	10	0.0411	9	10
10	9	0.0253	10	0.0664	9	10
max	16	0.1793	31	0.2172	10	31
avg	10.1	0.0552	14.9	0.1018	9.1	15

If we combine all information, we can say that the primary color ratio measure both spreads the distances over a bigger range (although the feature vectors have a lower dimension than those of the rainbow color ratio method) and puts the distortions of the same image closer to each other. So the primary color ratio measure can distinguish better between images that are distortions of each other and images that have a similar color scheme but a different shape.

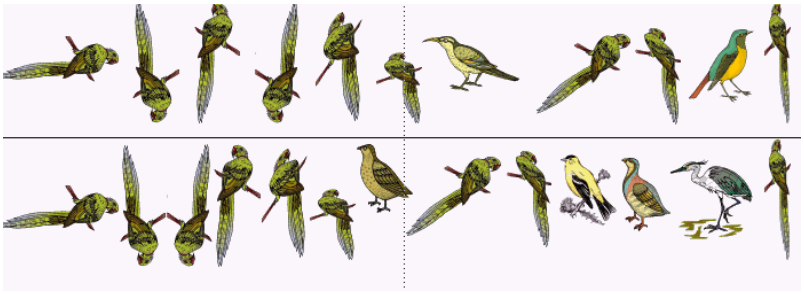

Figure 14. Comparison of the both color measures: closest images.

In Figure 14, we can see that the extra images found by both color methods are different. Images 7 and 10 of the first are different from images 7, 10, 11 or 12 from the second row. This leads to the novel idea of combining both the primary color ratio measure and the rainbow color measure by taking the intersection of the nearest neighbors found by the two methods. In the rainbow and primary color ratio measure, we still need to distinguish the relevant from the nonrelevant images in the output. This can be done either by the user of the system, or by some automatic method, for example an affine-invariant shape comparison method.

We first search the matching images by means of both the rainbow color ratio and the primary color ratio, and then reduce the output using the following simplification algorithm (Algorithm 7).

Algorithm 7 Simplification Algorithm

Step 1. Let L_1 be the output of the primary color ratio measure, containing m images. (The number m is the number of images returned that guarantees that 100% of the relevant images are found. In our previous experiments, $m = 31$ when using the rainbow color ratio method.) Let L_2 be the output of the rainbow color ratio measure, containing m images. Let L_1', L_2' and **Out** be lists of image ID's, initialize both lists to the empty list.

Step 2. Sequentially scan L_1 and L_2 in parallel. The first time, take 2 elements of L_2 such that we always read one element ahead in that list. Add each element that had been viewed to L_1' or L_2', corresponding to the list it originates from. Each time an element from both lists is read, check of L_1' and L_2' contain any common elements. If so, add the common element to **Out** and remove it from both L_1' and L_2'.

Step 3. Return **Out**.

In the Table above, we show again the number of elements we have to return in order to obtain 100% relevance. We can see that, except for image number 4 (i.e., in 90% of the cases), the first 9 elements of the intersection are the 9 distortions of the image (column 6). Column 7 shows how far we have to scan both lists before the number of intersections in column 6 of the corresponding row are found. This is the same as the maximum of the ranks for the primary and the rainbow color measure. The efficiency of the combination is significantly better than the efficiency of the primary color ratio method. We obtain that 98.9% of the returned images is relevant. In Figure 15, the performance of all three methods is combined into one graph.

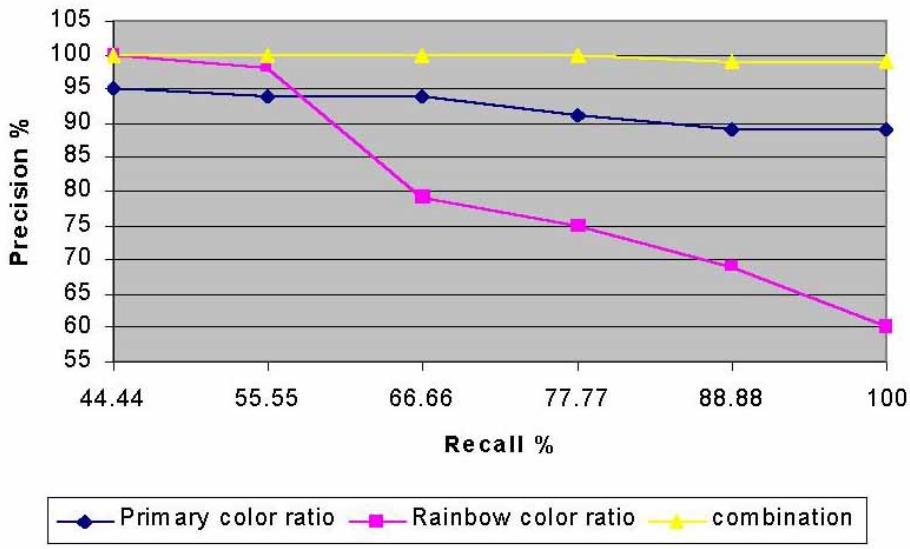

Figure 15. Comparison of the performance of the three color measures.

6.3.2. Comparing Shapes

As we already mentioned in the previous subsection, when the system returns the best matched to the user, either the user or some other technique needs to separate the relevant from the nonrelevant images. We now look if we can use the shape comparison method to do this separation. For the primary color ratio method we apply the shape comparison on the 16 nearest neighbors found by the primary color ratio method. For the rainbow color ratio method we search amongst the 31 nearest neighbors.

The following table shows the experimental results for the shape measure. We computed the distance for $N = 1$, i.e., the biggest polygon only is considered.

	Distance	Rainbow Color Rank	Primary Color Rank
1	0.1938	13	10
2	0.3986	16	11
3	0.4059	30	15
4	0.5903	29	10
5	0.5936	30	16
6	0.2126	10	10
7	0.2898	24	9
8	0.1999	23	12
9	0.7869	30	16
10	0.6930	31	15
max:	0.7869	31	16
avg:	0.4364	23.6	12.4

It seems that the shape comparison cannot add any value to the rainbow color ratio and primary color ratio measures. There are two explanations for this. First, the primary color and rainbow color ratio measures perform very well. Secondly, the boundary detection method used (i.e., the Hough transform) is not very accurate.

Without more accurate boundary detection algorithms, it seems that the applications of the affine-invariant triangulation lie mainly in the field of constraint databases [20,21]. Here, the exact equations that define the boundary of figures are known.

7. Concluding Remarks

We adopted the hierarchical data model of Chomicki and Revesz [1] 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 application of our affine-invariant spatial triangulation method to image indexing and retrieval is described in Section 6.

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.

Acknowledgments: An extended abstract, containing few technical details and examples, appeared as [42]. An earlier version of the application part appeared as part of [27].

Author Contributions: The authors contributed equally.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Chomicki, J.; Revesz, P. A geometric framework for specifying spatiotemporal objects. In Proceedings of the 6th International Workshop on Temporal Representation and Reasoning, Orlando, FL, USA, 1–2 May 1999; IEEE Computer Society Press: Washington, DC, USA, 1999; pp. 41–46.
2. Böhlen, M.H.; Jensen, C.S.; Scholl, M. *Spatio-Temporal Database Management, International Workshop STDBM 1999*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 1999.

3. Chen, C.X.; Zaniolo, C. SQLST: A spatio-temporal data model and query language. In Proceedings of the 19th International Conference on Conceptual Modeling (ER'00), Salt Lake City, UT, USA, 9–12 October 2000; Lecture Notes in Computer Science; Storey, V.C.; Laender, A.H.F.; Liddle, S.W., Eds; Springer: Berlin/Heidelberg, Germany, 2000; Volume 1920, pp. 96–111.
4. Chomicki, J.; Haesevoets, S.; Kuijpers, B.; Revesz, P. Classes of spatiotemporal objects and their closure properties. *Ann. Math. Artif. Intell.* **2003**, *39*, 431–461.
5. Frank, A.; Grumbach, S.; Güting, R.; Jensen, C.; Koubarakis, M.; Lorentzos, N.; Manopoulos, Y.; Nardelli, E.; Pernici, B.; Schek, H.-J.; et al. Chorochronos: A research network for spatiotemporal database systems. *SIGMOD Rec.* **1999**, *28*, 12–21.
6. Grumbach, S.; Rigaux, P. ; Segoufin, L. Spatio-temporal data handling with constraints. In Proceedings of the 6th International Symposium on Advances in Geographic Information Systems (ACM-GIS'98), Washington, DC, USA, 6–7 November 1998.
7. Güting, R.H.; Bohlen, M.H.; Erwig, M.; Jensen, C.S.; Lorentzos, N.A.; Schneider, M.; Vazirgiannis, M. A foundation for representing and querying moving objects. *ACM Trans. Databases Syst.* **2000**, *25*, 1–42.
8. Kuijpers, B.; Paredaens, J. ; Van Gucht, D. Towards a theory of movie database queries. In Proceedings of the 7th International Workshop on Temporal Representation and Reasoning, Cape Breton, NS, Canada, 7–9 July 2000.
9. Pfoser, D.; Tryfona, N. Requirements, definitions and notations for spatiotemporal application environments. In Proceedings of the 6th International Symposium on Advances in Geographic Information Systems (ACM-GIS'98), Washington, DC, USA, 6–7 November 1998.
10. Kuijpers, B.; Haesevoets, S. Closure properties of classes of spatio-temporal objects under boolean set operations. In Proceedings of the 7th International Workshop on Temporal Representation and Reasoning, Cape Breton, NS, Canada, 7–9 July 2000.
11. Erwig, M.; Schneider, M. Partition and conquer. In Proceedings of the 3rd International Conference on Spatial Information Theory, Laurel Highlands, PA, USA, 15–18 October 1997; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 1997; Volume 1329, pp. 389–408.
12. Geerts, F.; Haesevoets, S.; Kuijpers, B. A theory of spatio-temporal database queries. In Proceedings of the 8th International Workshop Database Programming Languages, DBPL 2001, Frascati, Italy, 8–10 September 2001.
13. Gyssens, M.; Van den Bussche, J.; Van Gucht, D. Complete geometric query languages. *J. Comput. Syst. Sci.* **1999**, *58*, 483–511.
14. Paredaens, J.; Van den Bussche, J.; Van Gucht, D. Towards a theory of spatial database queries. In Proceedings of the 13th ACM Symposium on Principles of Database Systems, Minneapolis, MN, USA, 24–26 May 1994.
15. Roberts, L. *Optical and Electro-Optical Information Processing*; Tippet, J., Ed.; Massachusetts Institute of Technology Press: Cambridge, MA, USA, 1965.
16. Hagedoorn, M.; Veldkamp, R.C. Reliable and efficient pattern matching using an affine invariant metric. *Int. J. Comput. Vis.* **1999**, *31*, 203–225.
17. Huttenlocher, D.P.; Klauderman, G.A.; Rucklidge, W. Comparing images using the Hausdorff distance. *IEEE Trans. Pattern Anal. Mach. Intell.* **1998**, *15*, 850–863.
18. Lamdan, J.S.Y.; Wolfson, H. Affine-invariant model-based object recognition. *Affine-Invariant Model-Based Object Recogn.* **1990**, *6*, 578–589.
19. Nielson, G. A characterization of an affine invariant triangulation. In *Geometric Modelling, Computing Supplementum*; Farin, G., Hagen, H., Noltemeier, H., Eds.; Springer: Berlin, Germany, 1993; Volume 8, pp. 191–210.
20. Paredaens, J.; Kuper, G.; Libkin, L. *Constraint Databases*; Springer: Berlin/Heidelberg, Germany, 2000.
21. Revesz, P. *Introduction to Constraint Databases*; Springer: Berlin/Heidelberg, Germany, 2002.
22. Vieira, T.; Martinez, D.; Andrade, M.; Lewiner, T. Estimating affine-invariant structures on triangle meshes. *Comput. Graph.* **2016**, *60*, 83–92.
23. Liu, Z.; Wang Y.; Jing, Y. A Feature Matching Algorithm Based on an Illumination and Affine Invariant for Aerial Image Registration. *Sens. Transducers* **2014**, *163*, 82–89.
24. Dou, J.; Li, J. Image matching based local Delaunay triangulation and affine invariant geometric constraint. *Optik* **2014**, *125*, 526–531.

25. Hoff, D.J.; Olver, P.J. Extensions of Invariant Signatures for Object Recognition. *J. Math. Imaging Vis.* **2013**, *45*, 176–185.
26. Collins, G. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Automata Theory and Formal Languages*; Lecture Notes in Computer Science; Brakhage, H., Ed.; Springer: Berlin/Heidelberg, Germany, 1975; Volume 33, pp. 134–183.
27. Haesevoets, S.; Kuijpers, B.; Revesz, P.Z. Efficient Affine-Invariant Similarity Retrieval. In Proceedings of the 2nd International Conference Geometric Modelling and Imaging (GMAI'07), Zurich, Switzerland, 4–6 July 2007; Sarfraz, M., Banissi, E., Eds.; IEEE Press: Hoboken, NJ, USA, 2007; pp. 99–108.
28. Bochnak, J.; Coste, M.; Roy, M.F. Real Algebraic Geometry. In *Ergebnisse der Mathematik und ihrer Grenzgebiete*; Springer: Berlin/Heidelberg, Germany, 1998; Volume 36.
29. De Berg, M.; van Kreveld, M.; Overmars, M.; Schwarzkopf, O. *Computational Geometry: Algorithms and Applications*; Springer: Berlin/Heidelberg, Germany, 2000.
30. De Berg, M.; van Kreveld, M.; Overmars, M.; Schwarzkopf, O. Arrangements and duality. In *Computational Geometry: Algorithms and Applications*; Springer: Berlin/Heidelberg, Germany, 2000; Chapter 8, pp. 165–182.
31. De Berg, M.; van Kreveld, M.; Overmars, M.; Schwarzkopf, O. Line segment intersection. In *Computational Geometry: Algorithms and Applications*; Springer: Berlin/Heidelberg, Germany, 2000; Chapter 2, pp. 18–43.
32. De Berg, M.; van Kreveld, M.; Overmars, M.; Schwarzkopf, O. Computational geometry. In *Computational Geometry: Algorithms and Applications*; Springer: Berlin/Heidelberg, Germany, 2000; Chapter 1, pp. 1–17.
33. Edelsbrunner, H.; Guibas, L.J.; Stolfi, J. Optimal point location in a monotone subdivision. *SIAM J. Comput.* **1986**, *15*, 317–340.
34. Dumortier, F.; Gyssens, M.; Vandeurzen, L.; Van Gucht, D. 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, Tucson, AZ, USA, 12–14 May 1997.
35. Novak, E.; Ritter, K. Some complexity results for zero finding for univariate functions. *J. Complex.* **1993**, *9*, 15–40.
36. Guttman, A. R-trees: A dynamic index structure for spatial searching. In Proceedings of the 1984 ACM SIGMOD international conference on Management of data, Boston, MA, USA, 18–21 June 1984; pp. 47–57.
37. Beckmann, N.; Kriegel, H.; Schneider, R.; Seeger, B. The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles. In Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data, Atlantic City, NJ, USA, 23–25 May 1990; pp. 322–331.
38. Kamel, I.; Faloutsos, C. Hilbert R-tree: An Improved R-tree using Fractals. In Proceedings of the 20th International Conference on Very Large Data Bases VLDB'94, Santiago de Chile, Chile, 12–15 September 1994; pp. 500–509.
39. Finkel, R.A.; Bentley, J.L. Quad Trees: A Data Structure for Retrieval on Composite Keys. *Acta Inform.* **1974**, *4*, 1–9.
40. Canny, J. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.* **1986**, *8*, 679–698.
41. Trucco, E.; Verri, A. *Introductory Techniques for 3-D Computer Vision*; Prentice Halli, Inc.: Upper Saddle River, NJ, USA, 1998.
42. Haesevoets, S.; Kuijpers, B. Time-dependent affine triangulation of spatio-temporal data. In Proceedings of the 12th ACM International Workshop on Geographic Information Systems (ACM-GIS'04), Washington, DC, USA, 12–14 November 2004.