RESEARCH ARTICLE

Approximating the evolution of rotating moving regions using bezier curves

José Duarte[a], Paulo Dias[a,b,c] and José Moreira[a,b,c]

[a]Institute of Electronics and Informatics Engineering of Aveiro (IEETA), [b]Department of Electronics, Telecommunications and Informatics (DETI), [c]Intelligent Systems Associate Laboratory (LASI), University of Aveiro, Campus Universitário de Santiago 3810-193, Aveiro, Portugal

ARTICLE HISTORY
Compiled November 24, 2022

ABSTRACT
The region interpolation methods proposed in the moving objects databases literature impose restrictions that can have a significant impact on the representation of the evolution of moving regions, in particular, when a rotation occurs between two observations. In this paper, we propose a data model for moving regions that allows moving segments to rotate and change their length during their evolution between two observations and uses quadratic Bezier curves to define the trajectories of their endpoints. This introduces a new class of moving regions called rotating moving regions (*rmregion*s). We present algorithms for operations involving *rmregion*s and we propose a strategy to allow different interpolation methods to be used in the context of moving objects databases by approximating the interpolations they create using *rmregion*s. We demonstrate our strategy using a reference implementation and compare results obtained when using the strategy presented here and the region interpolation methods and the spatiotemporal operations proposed in the state-of-the-art. Experimental results show that our strategy can be used to complement the region interpolation methods proposed in the moving objects databases literature.

KEYWORDS
region interpolation problem; moving regions; moving objects databases; spatiotemporal operations; interpolation methods

1. Introduction

Moving objects databases (Güting and Schneider 2005) allow representing and querying the evolution of objects whose position, shape and extent can change continuously over time. These objects are called moving points when only the evolution of the position is relevant and moving regions when the evolution of the position, shape and extent are relevant. Moving regions can represent the evolution of real-world phenomena, e.g. an iceberg, a forest fire and biological cells, and are created from observations of the evolution of the phenomena being represented. The evolution between two observations is created using a region interpolation method.

CONTACT José Duarte. Email: hfduarte@ua.pt
CONTACT Paulo Dias. Email: paulo.dias@ua.pt
CONTACT José Moreira. Email: jose.moreira@ua.pt

The region interpolation methods proposed in the literature impose the restriction that moving segments are not allowed to rotate between two observations (Tøssebro and Güting 2001, McKenney and Webb 2010, McKenney and Frye 2015, Heinz and Güting 2016), or moving segments are allowed to rotate but moving regions have fixed shapes during their evolution between two observations (Heinz and Güting 2018). The restriction imposed in the first case allows known efficient algorithms to be used to develop operations involving moving regions. However, a consequence of this restriction is that a good representation of the evolution of the phenomena is not always generated when a rotation occurs between two observations (Heinz and Güting 2018). A possible solution is to collect observations at sub-intervals between the two observations, to better approximate the rotation. This, however, can increase the size of the moving regions and the execution time of spatiotemporal operations considerably (Heinz and Güting 2018), since a larger number of observations is used to create the moving regions and spatiotemporal operations have to be performed on all sub-intervals. The restriction imposed in the second case dictates that rotations can be represented accurately but the shape of moving regions cannot change between two observations.

Other interpolation methods, e.g. (Alexa *et al.* 2000, Baxter *et al.* 2008), have been proposed in the literature to represent the continuous transformation between two geometries and their main goal is to obtain a natural transformation of the geometry. However, in general, these methods create moving regions whose moving segments are allowed to rotate and change their length during their evolution between two observations and use different formulas with different levels of complexity to define the trajectories of the vertices of the moving regions. As a consequence, algorithms to implement operations involving these moving regions are inefficient.

We would like to be able to use different interpolation methods to create moving regions, in particular, in cases where the region interpolation methods proposed in the moving objects databases literature do not give good results and develop algorithms for spatiotemporal operations that are independent, and have a complexity that is also independent, of the interpolation method used. To achieve this, we propose a discrete data model to represent moving regions, based on the data model presented in (Güting *et al.* 2000, Forlizzi *et al.* 2000), that uses quadratic Bezier curves to define the trajectories of the vertices of the moving regions and allows moving segments to rotate and change their length during their evolution between two observations. This introduces a new class of moving regions called Rotating Moving Regions (*rmregion*s). We develop algorithms for the following fundamental operations involving *rmregion*s: *atinstant* gives the value of a *rmregion* at a specified instant in time, *interpolate* creates a *rmregion*, *intersects* gives the time intervals where a moving point and a *rmregion* and two *rmregion*s intersect. Then, the interpolations created by different interpolation methods are approximated using (are projected to) *rmregion*s.

Experimental results show that this strategy can be used to complement the region interpolation methods proposed in the moving objects databases literature in cases where they do not create a good representation of the evolution of a phenomenon. The quality of an interpolation and a natural interpolation are, we believe, case (application) dependent and providing a definition for these concepts requires the cooperation of experts and practitioners in various fields. Our goal is to allow various interpolation methods to be used in the context of moving objects databases and let the user choose and use the one that provides the best results according to the user's needs.

This paper is organized as follows. The next two sections present related work, data types, concepts and an overview relevant for this paper. Then, a discrete data model for *rmregion*s is introduced. After that, we propose primitives to develop operations involving *rmregion*s and algorithms for some operations. Following that, we introduce a procedure to create *rmregions*. Then, we present experimental results obtained using the strategy and the operations developed. This includes a comparison with results obtained using the region interpolation methods proposed in the literature. The conclusions and guidelines for future work end the paper.

2. Related work

(Güting *et al.* 2000) and (Forlizzi *et al.* 2000) propose a data model for moving objects databases that includes data types and operations to represent and query the evolution of moving regions. Moving regions are defined using moving segments and their evolution is represented using the *sliced representation* (See Section 3 for more details). This is the most well-known data model for moving objects databases and was used as a reference for other data models, e.g. the CMR model for moving regions (McKenney *et al.* 2014) and the model for moving regions of fixed shape (Heinz and Güting 2018), and several works presented in the moving objects databases literature. With the exception of the model for moving regions of fixed shape, these models and works impose the restriction that moving segments are not allowed to rotate during their evolution between two observations. This allows the use of known efficient algorithms to implement spatiotemporal operations and is a trade-off between the efficiency of spatiotemporal operations involving moving regions and the representation of the evolution of the phenomena. Efficient region interpolation methods based on the rotating plane algorithm (Tøssebro and Güting 2001) exist to create moving regions from observations, following this restriction: (Tøssebro and Güting 2001), (McKenney and Webb 2010), (McKenney and Frye 2015) and (Heinz and Güting 2016). This however, can have a significant impact on the representation of the evolution of the phenomena, in particular, when a rotation occurs between two observations. More recently, an alternative model for moving regions, called the polyhedra based model, that uses polyhedra to represent moving regions instead of the *sliced representation* was proposed in (Heinz and Güting 2020). In this model, moving regions are created by considering observations as top and bottom surfaces of a polyhedron and by using the rotating plane algorithm to create the lateral surface of the corresponding polyhedron. As a consequence, the representation of the evolution of a moving region created using this model is similar to the representation created when imposing the restriction that moving segments cannot rotate between two observations. The model for moving regions of fixed shape allows moving segments to rotate, but imposes the restriction that their length is fixed during their evolution between two observations.

Other interpolation methods exist that allow moving segments to rotate and change their length during their evolution between two observations, i.e. during interpolation, e.g. rigid interpolation methods (Alexa *et al.* 2000, Baxter *et al.* 2008, Liu *et al.* 2011) and interpolation methods based on physical principles (Yan *et al.* 2004). These methods have the potential to create a good or alternative representation of the evolution of real-world phenomena, but (i) the trajectories of the vertices during interpolation can be defined using different functions with different levels of complexity, (ii) there is not a method that creates the best representation in all cases and (iii) the 3D representation of the moving regions created using these methods are objects with arbitrary shape possibly containing curved surfaces. This can make algorithms for spatiotemporal operations involving moving regions created using these methods very inefficient and demand the use of specific data types.

In this work we allow moving segments to rotate and change their length during their evolution between two observations and present data types and operations to represent and query the evolution of the corresponding new class of moving regions.

3. Background

This section presents data types used in moving objects databases, proposed in (Forlizzi *et al.* 2000), using a discrete model, a more detailed discussion about region interpolation methods and a discussion on the use of interpolation methods in the context of moving objects databases. In the following, when presenting snapshots of an interpolation, snapshots depicted with a blue background color are the observations or the samples (when using Bezier curves) used to create the interpolation.

3.1. *Moving Objects Databases Data Types*

Bool, Instant and Interval. A *bool* can have one of the following values: *true*, *false* and *undefined.* An *instant* is a value in the real numbers that represents an instant in time. An *interval* is represented by the quadruple (t_b, t_e, l, r) where $t_b \leq t_e$, l and r indicate if t_b and t_e belong to (are included) in the interval, respectively, and t_b and t_e are *instant* types.

Point and Region. A *point* is defined by the tuple (x, y), i.e. its coordinates in 2D-space. A *region* is defined as a set of disjoint faces that can touch at an isolated point, but cannot have overlapping boundary segments. A face can have zero or more holes and a hole can have zero or more faces. A hole must be inside a face. A face and a hole can touch at an isolated point. Faces and holes are represented by simple polygons. A *point* can be inside, on the border (touch) or outside a *region.*

Sliced Representation. The *sliced representation* (See Figure 1), proposed in (Forlizzi *et al.* 2000), decomposes the evolution of an object into fragments called slices (or units). A unit has (i) a start and end values, these are usually consecutive observations of the actual evolution of the object, (ii) a time interval during which the unit is defined and (iii) a function that describes the evolution of the object between the start and end values (or observations). A region interpolation method is a function used to describe the continuous evolution of a region during a unit. These functions are discussed in more detail in Section 3.2.

Moving Objects. Moving objects represent the continuous evolution of objects over time and are constructed using the *sliced representation.* That is, moving objects are constructed using units defined at disjoint time intervals. In general, a moving object has several units. The *atinstant* operation on a moving object gives the value (the state or the representation) of the moving object at a specific instant in time.

Moving Bool. A *moving bool* (*mbool*) is a *bool* whose value can change over time. A *unit bool* (*ubool*) represents the value of a *mbool* during a time interval, i.e. *ubool* : $(bool, i)$ where $i \in Interval$ is the time interval during which the *ubool* is defined and *bool* is the value that it assumes during i.

Moving Point. A *unit point* (*upoint*) describes the evolution of the position of a point between two known positions during a time interval, i.e. *upoint* : $(S, \vec{v}, i)$, where S is the starting position, $\vec{v}$ is a vector that describes the movement of the point and $i \in Interval$ is the time interval during which the unit is defined. The position of the point at $t \in i$ is given by the parametric functions: $x(t) := S_x + t^N \cdot \vec{v}_x$ and $y(t) := S_y + t^N \cdot \vec{v}_y$, where t^N represents t normalized to the interval $[0, 1]$. A *moving point* (*mpoint*) is a set of *upoint*s with disjoint time intervals.

Moving Region. A *moving region* (*mregion*) is a region whose position, shape and extent can change continuously over time. A *unit region* (*uregion*) describes the evolution of a *mregion* during a time interval between two observations, i.e. *uregion* : $(i, F \subset MFace)$, where F is a set of moving faces and i is the time interval during which the *uregion* is defined. A moving face is represented using closed moving polygon cycles formed by moving segments, that represent the face and its holes. A *moving segment* (*mseg*) is defined as a tuple *mseg* : $(s_b, s_e), s_b, s_e \in Seg$, where Seg is the set of all segments in 2D-space, s_b and s_e are parallel and s_b or s_e but not both can be degenerated to a point. s_b and s_e represent the initial and final configurations of the *moving segment*. The evolution of the *moving segment* between s_b and s_e is computed using linear interpolation. As a consequence, all the in-between configurations of a *moving segment* at a time instant are segments parallel to s_b and s_e and a rotating segment has to be simulated using two *mseg*s (See Figure 1 (A)). This restriction allows the use of efficient algorithms to develop operations involving *mregions*. A *mregion* is a set of *uregion*s with disjoint time intervals.

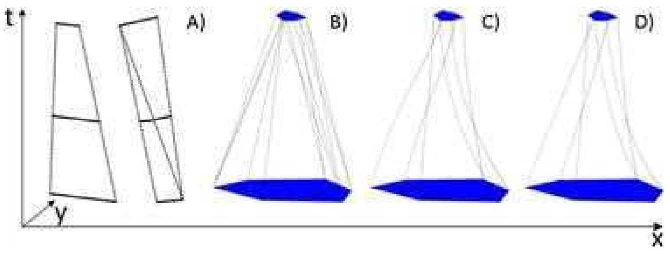

Figure 1. Evolution of two moving segments between two observations (A) and the trajectories of the endpoints of the moving segments of a moving region during a unit (B), created using the classical model for moving segments. Trajectories of the endpoints of the moving segments created using the rigid interpolation method proposed in (Alexa *et al.* 2000, Baxter *et al.* 2008) (C) and the strategy presented in this paper (D), i.e. (D) is an approximation of (C) using quadratic Bezier curves.

3.2. *Region Interpolation Methods*

Region interpolation methods are used to create the evolution of a region between two observations. We consider that currently there are two main models for moving segments, used by the region interpolation methods proposed in the moving objects databases literature. We call these: the classical model of moving segments, used in (Tøssebro and Güting 2001, McKenney and Webb 2010, McKenney and Frye 2015, Heinz and Güting 2016, 2020), and the model for fixed moving segments, used in Heinz and Güting (2018). When making this distinction we are emphasizing the fact that in the first and in the latter moving segments are defined with different restrictions.

3.2.1. *Modeling the Evolution of Moving Segments*

A consequence of the restriction imposed in the classical model of moving segments is that a good representation of the evolution of a region is not always generated (Moreira *et al.* 2016, Heinz and Güting 2018).

In (Heinz and Güting 2018) the authors observe that the classical model of moving segments does not give good results when representing the evolution of regions with fixed shape, in particular, when the two observations are rotational asymmetric and a large rotation exists between them (See Figure 2 (top)). However, this also occurs when representing the evolution of regions with a non-fixed shape (See Figure 2 (bottom)).

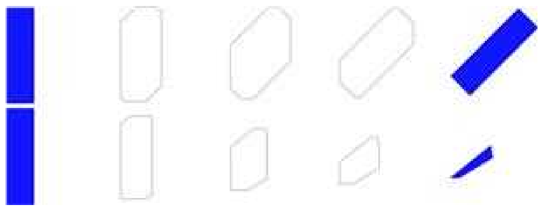
Figure 2. 5 snapshots of the evolution of a box with a fixed (top) and a non-fixed (bottom) shape between two observations, created using the classical model of moving segments.

If more observations of the evolution of the box are taken between the two observations, then, the interpolation created using the classical model of moving segments approximates the actual evolution of the box, but this can increase the size (the number of units) of the moving regions and the execution time of the spatiotemporal operations considerably (Heinz and Güting 2018).

3.2.2. *Modeling the Evolution of Concavities*

Region interpolation methods, except the model for fixed moving regions, do not always create a good representation of the evolution of concavities, i.e. a concavity can simultaneously disappear to and appear from a point during its evolution between two observations. In Figure 3, the large concavity on top of the iceberg simultaneously disappears to and appears from a point. This is because concavities are matched using matching strategies, e.g. the overlap amount and the distance between the concavities in the two observations. In (Heinz and Güting 2016) the user can provide customized matching strategies. However, matching strategies can be case-dependent and it could even be the case that a matching strategy works for a concavity but not for another concavity on the same region.

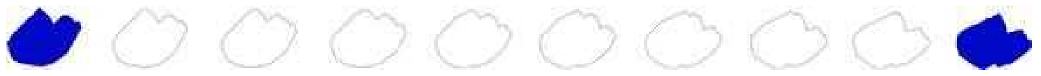
Figure 3. 8 snapshots of the evolution of an iceberg between two consecutive observations, created using the method proposed in (Heinz and Güting 2016) with an overlap matching strategy.

3.3. ***Alternative Region Interpolation Methods***

The discussion presented in Subsections 3.2.1 and 3.2.2 suggests the use of alternative region interpolation methods to handle cases where the region interpolation methods proposed in the moving objects databases literature do not provide a good (or acceptable) representation of the evolution of moving regions over time. The model for fixed moving regions (Heinz and Güting 2018) is a good example of this argument.

Several interpolation methods have been proposed in the literature to create the continuous transformation between two geometries. Figure 4 shows snapshots of the evolution of the box and the iceberg presented in Subsections 3.2.1 and 3.2.2, created using the rigid interpolation method proposed in (Alexa *et al.* 2000, Baxter *et al.* 2008).

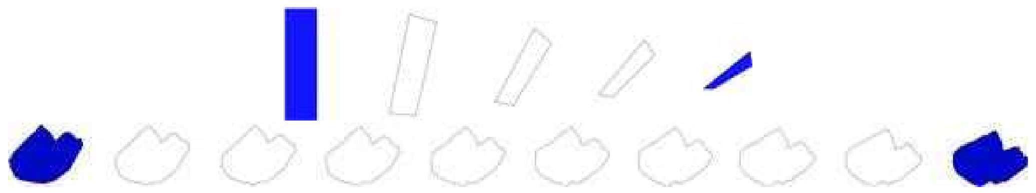

Figure 4. Snapshots of the evolution of a box (top) and an iceberg (bottom) between two consecutive observations, created using the rigid interpolation method proposed in (Alexa *et al.* 2000, Baxter *et al.* 2008).

The method used in the examples shown here creates moving regions whose moving segments are allowed to rotate and change their length during the interpolation between two observations and can represent rotations $> 2\pi$ if the user specifies additional parameters (See Figure 5). A one-to-one correspondence between the vertices of the source and target geometries must be given.

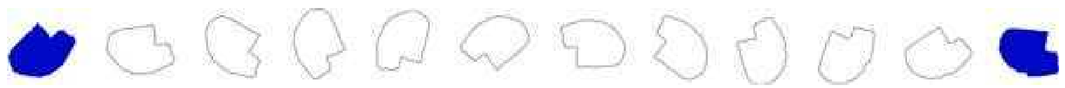

Figure 5. 10 snapshots of the evolution of an iceberg that rotates 410 degrees between two observations, created using the rigid interpolation method proposed in (Alexa *et al.* 2000, Baxter *et al.* 2008).

However, there isn't an interpolation method that generates the best representation in all cases and different interpolation methods can use functions with different characteristics and complexity to define the trajectories of the vertices of the moving regions during interpolation. As a consequence of the latter, algorithms developed to implement spatiotemporal operations involving moving regions may work only with some interpolation method(s), or may be particularly inefficient when using specific interpolation methods. Ideally, we would like to be able to use different interpolation methods and develop algorithms for spatiotemporal operations that are independent, and have a complexity that is also independent, of the interpolation method used to create moving regions. Also, some interpolation methods can create a good representation of the evolution of moving regions, but are too inefficient or complex to be used directly to develop spatiotemporal operations. In this work we propose a strategy that uses interpolation methods as black boxes to allow different interpolation methods to be used in the context of moving objects databases. This allows creating a good approximation of rotations using fewer observations. The strategy uses the *sliced representation* to represent the evolution of moving regions.

4. Discrete Model for Rotating Moving Regions

A *rmregion* is similar to a classical moving region (*mregion*). The difference is that a *rmregion* does not impose the restriction that moving segments cannot rotate during their evolution between two observations and the trajectories of their endpoints are defined using quadratic Bezier curves. We start by introducing the type *curve*.

$curve : (cp_0, cp_1, cp_2)$
where $cp_0, cp_1, cp_2 \in Point$ are the control points of the curve that is defined in the interval $[0, 1]$. We use a new data type to represent the moving segments of a *rmregion*, that we call *rotating moving segment* (*rmseg*) and that is defined as:
$rmseg : (curve_1, curve_2), curve_1, curve_2 \in Curve$
where *Curve* is the set of all *curve* types and $curve_1$ and $curve_2$ are the curves that give the trajectories of the endpoints of the *rmseg*. The initial and final configurations of a *rmseg* may or may not be parallel and one of them, but not both, can be degenerated to a point (See Figure 6 (A and C)). The position of an endpoint of a *rmseg* is given by the parametric equations:

$$\begin{aligned} x(\tau) &= cp0_x(1-\tau)(1-\tau) + 2cp1_x(1-\tau)\tau + cp2_x\tau^2 \\ y(\tau) &= cp0_y(1-\tau)(1-\tau) + 2cp1_y(1-\tau)\tau + cp2_y\tau^2 \end{aligned} \quad (1)$$

where τ represents a time instant normalized to $[0, 1]$ and cp_0, cp_1 and cp_2 are the control points of the curve that defines the trajectory of the endpoint. Then, a moving face is represented using closed moving polygon cycles formed by *rmseg*s, that represent the face and its holes. We define the set of all moving cycles and moving faces as:
$MCycle = \{\{rs_0, \ldots, rs_{\varrho-1}\} \mid \varrho \geq 3, \forall\chi \in \{0, \ldots, \varrho - 1\} : rs_\chi \in RMSseg\}$
$MFace = \{(c, H) \mid c \in MCycle, H \subset MCycle\}$
where *RMSseg* is the set of all *rmseg*s. A *unit rmregion* describes the evolution of a *rmregion* during a time interval between two observations and is defined as: $urmregion : ([t_b, t_e], F \subset MFace)$, such that $\forall t \in [t_b, t_e] : evaluate(F, t) \in Region$, where F is a set of moving faces, $[t_b, t_e]$ is the time interval during which the *urmregion* is defined, *Region* is the set of all *region*s as defined in (Forlizzi *et al.* 2000) and *evaluate* gives the representation of F at t. A *rmregion* is a set of *urmregions* with disjoint time intervals, i.e. $rmregion \subset URMRegion$, where $URMRegion$ is the set of all *urmregion*s. *rmregion*s are arbitrary shapes with curved surfaces, when viewed in 3D-space (See Figure 6 (B)).

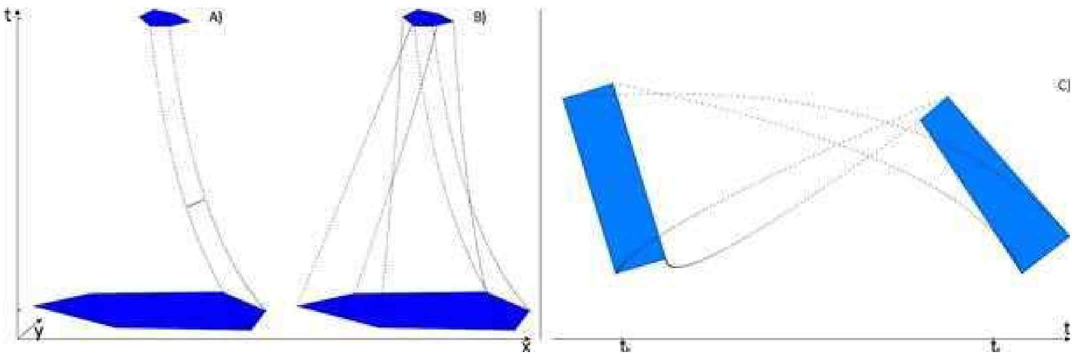

Figure 6. Evolution of a *rmseg* between two configurations (A). Trajectories of the *rmseg*s of a *rmregion* during a *urmregion* (B) and (C).

The data model for *rmregion*s and the data model for *mregion*s can be combined, but we chose not to combine them because efficient algorithms for some operations involving *rmregion*s still have to be developed, e.g. the *traversedarea* operation, and our strategy to create *rmregion*s currently does not provide explicit mechanisms to detect and handle self-intersections. Also, the use of *rmregion* makes clear that it refers to a new class of moving regions.

Using quadratic Bezier curves is a trade-off between the number of functions needed to obtain a good approximation of a rotation between two observations and the efficiency of spatiotemporal operations.

5. Implementation of Operations Involving Rotating Moving Regions

This section presents some fundamental operations for *rmregions*. The algorithms apply with the following considerations: the moving objects involved have 1 unit and *rmregions* have 1 simple face. In the general case the algorithms have to be iterated for each unit and face.

5.1. *Atinstant: rmregion* × *instant* → *region*

Atinstant gives the representation of a *rmregion* at a specific instant of time t or $\perp$ (undefined) if the *rmregion* is not defined at t. The representation of a *rmregion* at t is obtained by computing the positions of the endpoints of its moving segments at t using the parametric functions in Equation (1) and constructing the corresponding *region* in 2D-space. Considering that the *urmregion* that contains t can be found in $O(log\, k)$, where k is the number of *urmregions* of the *rmregion*, using binary search, the complexity of *atinstant* is $O(log\, k+m)$, where m is the number of moving segments of the *rmregion*.

5.2. *Intersects: mpoint* × *rmregion* → *mbool*

First, we define a primitive to find the instants of time and the time intervals where a moving point and a moving segment intersect. Then, this primitive is used to develop an algorithm for *intersects* : *mpoint* × *rmregion* → *mbool*.

5.2.1. Moving Point × Moving Segment

Given *Ax(t), Ay(t), Bx(t)* and *By(t)*, the quadratic Bezier curves that define the trajectories of the endpoints of a moving segment *ms*, given by the parametric equations in Equation (1) and *Px(t)* and *Py(t)*, the curves that define the trajectory of a moving point *mp* in x and y, where, without loss of generality, *ms* and *mp* are defined during an interval of time $i = [t_b, t_e]$, the instants when *ms* and *mp* are collinear can be found using the following polynomial:

G(t) = Ax(t) (By(t) - Py(t)) + Bx(t) (Py(t) - Ay(t)) + Px(t) (Ay(t) - By(t)).

Where t represents real-world time. The trajectory of *mp* can be linear or quadratic. As a consequence, *G(t)* is a quartic polynomial. Then, the instants of time where *ms* and *mp* intersect and touch are given by:

$$T = \{t : G(t) = 0 \land t \in i \ \land \ \textit{on-segment}(\textit{atinstant}(mp, t), \textit{atinstant}(ms, t))\}.$$

Where *on-segment* ensures that the roots found correspond to instants of time where *mp* is actually on *ms*. When using this primitive to develop operations involving a moving point and a *rmregion*, a moving segment represents a part of the boundary of the *rmregion*. As a consequence, this primitive finds the instants of time where the moving point touches the *rmregion*.

5.2.2. Reference Implementation

Intersects (See Algorithm 1) is implemented using the primitive defined in Section 5.2.1 and can be easily parallelized. *Intersects* gives the time intervals where a moving point and a *rmregion* intersect (See Figure 7).

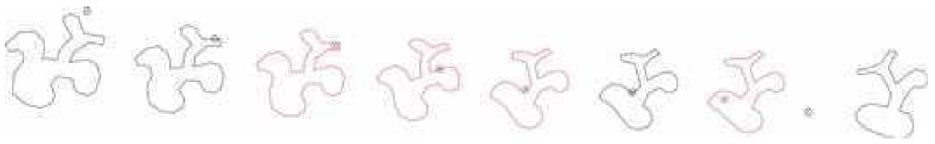

Figure 7. 8 snapshots of the *intersects* operation involving a moving point and a *rmregion*. Snapshots depicted in red correspond to instants where the two objects intersect.

Algorithm 1 Generic algorithm to find the time intervals where a moving point and a *rmregion intersect.*

Input: A moving point and a rotating moving region.
Output: A moving bool whose value is true when the two moving objects intersect.

```
1:  procedure INTERSECTS($mp, mr$)
2:      $mbool \leftarrow \emptyset$
3:      $i \leftarrow deftime(mp) \cap deftime(mr)$
4:      $it \leftarrow []$
5:      for $ms \in mr$ do
6:          $add(find\text{-}intersection\text{-}times(ms, mp, i), it)$
7:      $sort(it)$
8:      $\mu \leftarrow length(it)$
9:      for $j = 0$ to $j < \mu - 1$ do
10:         $t \leftarrow (it_j + it_{j+1})/2$
11:         if $inside(atinstant(mp, t), atinstant(mr, t))$ then
12:             $add(ubool(true, [it_j, it_{j+1}]), mbool)$
13:         else
14:             $add(ubool(true, [it_j, it_j]), mbool)$
15:             $add(ubool(false, ]it_j, it_{j+1}[), mbool)$
16:             $add(ubool(true, [it_{j+1}, it_{j+1}]), mbool)$
17:     if $inside(atinstant(mp, min(i)), atinstant(mr, min(i)))$ then
18:         $add(ubool(true, [min(i), it_0]), mbool)$
19:     else
20:         $add(ubool(false, [min(i), it_0[), mbool)$
21:     if $inside(atinstant(mp, max(i)), atinstant(mr, max(i)))$ then
22:         $add(ubool(true, [it_{\mu-1}, max(i)]), mbool)$
23:     else
24:         $add(ubool(false, ]it_{\mu-1}, max(i)]), mbool)$
25:     return $mbool$
```

INTERSECTS(), presented in Algorithm 1, receives a moving point and a *rmregion*. We check when the two moving objects are defined simultaneously (line 3). We find the intersection times between the two moving objects by iterating through all the moving segments of the *rmregion* and using the primitive presented in Section 5.2.1 (lines 5 and 6), this has complexity $O(m)$, where m is the number of moving segments, and sort the intersection times found (line 7), this has complexity $O(n\,log\,n)$, where n is the number of intersection times found. Then, we construct the respective *mbool* according to the intersection times found and the initial and final states of the two moving objects w.r.t each other. We consider that the two moving objects intersect when the moving point is on the boundary and inside the *rmregion*. *Intersects* can be used to implement other operations, e.g. *touches*, *disjoint*, *inside* and operations involving a *rmregion* and a *region*. The semantics of these operations can be different depending on the application. In practice, finding the intersection times dominates the execution time since the largest number of intersection times found during an interval is likely to be a small constant. If the two moving objects have k_1 and k_2 units then, we can traverse the units of the two objects in $O(k_1 + k_2)$. The complexity of the algorithm is therefore $O(k_1 + k_2 + M)$, where M is the number of moving segments in all the units of the *rmregion*. Efficient algorithms exist to determine if a point is inside a region, e.g. the partial-scan plumb-line algorithm (Güting and Ding 2004). *Intersects* can handle moving points with quadratic trajectories directly.

5.3. *Intersects: rmregion* × *rmregion* → *mbool*

We define a primitive to find the instants of time and the time intervals where two moving segments intersect, over time. Then, this primitive is used to develop an algorithm for $intersects : rmregion \times rmregion \rightarrow mbool$.

5.3.1. *Moving Segment* × *Moving Segment*

Given *Ax(t), Ay(t), Bx(t)* and *By(t)* and *Cx(t), Cy(t), Dx(t)* and *Dy(t)*, the quadratic Bezier curves that define the trajectories of the endpoints of two moving segments ms_1 and ms_2 in x and y, respectively, where, without loss of generality, ms_1 and ms_2 are defined in an interval of time $i = [t_b, t_e]$, the instants of time when the endpoints of one moving segment are collinear with the other moving segment can be found using the following polynomials:

$$G_1(t) = (Bx(t) - Ax(t))(Cy(t) - Ay(t)) - (By(t) - Ay(t))(Cx(t) - Ax(t)).$$
$$G_2(t) = (Bx(t) - Ax(t))(Dy(t) - Ay(t)) - (By(t) - Ay(t))(Dx(t) - Ax(t)).$$
$$G_3(t) = (Dx(t) - Cx(t))(Ay(t) - Cy(t)) - (Dy(t) - Cy(t))(Ax(t) - Cx(t)).$$
$$G_4(t) = (Dx(t) - Cx(t))(By(t) - Cy(t)) - (Dy(t) - Cy(t))(Bx(t) - Cx(t)).$$

Where t represents real-world time and $G_\alpha(t)$ are quartic polynomials. Then, the instants of time where ms_1 and ms_2 touch and meet are given by:

$$\begin{aligned} T = \{t : G_\alpha(t) = 0 \wedge t \in i \wedge (&\\ &touch(atinstant(ms_1, t), atinstant(ms_2, t)) \vee \\ &meet(atinstant(ms_1, t), atinstant(ms_2, t))), \ \alpha = 1, \ldots, 4\}. \end{aligned}$$

Where *touch* and *meet* ensure that the roots found correspond to instants of time where the two moving segments touch or meet. Two segments touch if one endpoint of one segment is in the interior of the other segment. Two segments meet if they have a common endpoint. To find the time intervals where the two moving segments intersect, we sort T to an ordered set T_S. Then, for each consecutive pair t_σ, $t_{\sigma+1} \in T_S$ we construct the following tuples: $([t_\sigma], true)$, I_σ, $([t_{\sigma+1}], true)$, where I_σ is given by rule R_σ and each tuple has an *interval* and a *bool* values associated with it. If t_b, $t_e \notin T_S$ we check if the two moving segments intersect at the two instants of time, as needed, and add the corresponding tuples constructed using rules R_b and R_e to the beginning and to the end of the list of tuples constructed from T_S, respectively.

$T_S = sort(T) = \langle t_0, t_1, \ldots, t_\mu \rangle$.

$$R_\sigma = \begin{cases} ([t_\sigma, t_{\sigma+1}], true) & \text{if } intersects(atinstant(ms_1, t), atinstant(ms_2, t)). \\ (]t_\sigma, t_{\sigma+1}[, false) & otherwise. \end{cases}$$

Such that $t = (t_\sigma + t_{\sigma+1})/2$.

$$R_b = \begin{cases} ([t_b, t_0], true) & \text{if } intersects(atinstant(ms_1, t_b), atinstant(ms_2, t_b)). \\ ([t_b, t_0[, false) & otherwise. \end{cases}$$

$$R_e = \begin{cases} ([t_\mu, t_e], true) & \text{if } intersects(atinstant(ms_1, t_e), atinstant(ms_2, t_e)). \\ (]t_\mu, t_e], false) & otherwise. \end{cases}$$

We construct the final result by observing that if the *bool* value associated with ρ consecutive tuples $u_\iota, u_{(\iota+1)}, \ldots, u_{(\iota+\delta)}$ does not change, then, the ρ tuples can be merged. The time intervals where the two moving segments intersect are given by the tuples with a *true* value. When using this primitive to develop operations involving two *rmregion*s, a moving segment represents a part of the boundary of a *rmregion.* As a consequence, when two moving segments intersect the two *rmregion*s intersect and when two moving segments touch the two *rmregion*s can touch or intersect, depending on how these relationships are defined.

5.3.2. Reference Implementation

Intersects gives the time intervals where two *rmregion*s intersect (See Figure 8). A generic algorithm to find the intersections between two *rmregion*s is similar to Algorithm 1 presented in Section 5.2. That is, we find the time interval i where the two *rmregion*s are defined simultaneously. Then, for every moving segment of one *rmregion* we find the time intervals where it intersects with the moving segments of the other *rmregion* during i, using the primitive defined in Section 5.3.1. This has complexity $O(m_1 m_2 + N\, log\, n_{max})$, where m_1 and m_2 are the number of moving segments of the two *rmregion*s, respectively, N is the total number of intersections found and n_{max} is the maximum number of intersections found between two moving segments. If no intersections are found we have to check if one of the *rmregion*s is inside the other. If this is the case the two *rmregion*s intersect during i. If intersections are found, we sort and merge them. This step has complexity $O(n\, log\, n)$, where n is the number of time intervals (intersections) found. If the endpoints of i are not included in the intersections found we check if one of the *rmregion*s is inside the other at the respective instants of time, as needed, and update the intersections found accordingly. Finally, we check if one of the *rmregion*s is inside the other between two time intervals where it is known that the two *rmregion*s intersect and merge the intervals where this condition is true. This procedure can be used to develop other operations, e.g. *inside*, *touches* and *overlaps.* Checking if one *rmregion* is inside the other has complexity $O(ls)$, where l is the number of times the operation is performed and s is the number of segments of the *rmregion* with the largest number of segments (assuming we check the inside condition by using a *point* inside *region* check). The overall complexity is therefore $O(m_1 m_2 + N\, log\, n_{max} + n\, log\, n + ls)$. If we change the primitive presented in Section 5.3.1 to consider that the endpoints of one of the moving segments follow linear functions, *intersects* can also be used to find the time intervals where a *rmregion* and a *mregion* intersect.

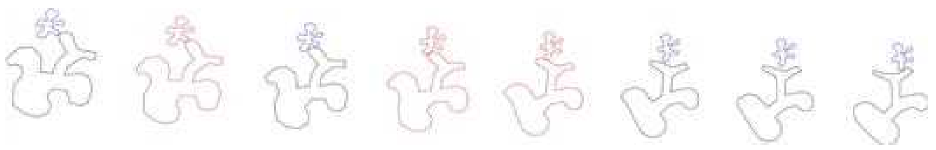

Figure 8. 8 snapshots of the *intersects* operation involving two *rmregions*. Snapshots depicted in red correspond to instants where the two *rmregions* intersect.

6. A Strategy to Use Different Interpolation Methods in the Context of Moving Objects Databases

Our goal is to allow the use of different interpolation methods in the context of moving objects databases. Because these methods can have different characteristics and complexity, as discussed previously, using them as black boxes has advantages. To achieve this, we project the interpolations created using these methods to a common representation that is then used. Thus, the algorithms to develop spatiotemporal operations become independent of the interpolation method used. This is a trade-off. The original interpolation is approximated, but the algorithms for spatiotemporal operations are, in general, more efficient than using the interpolation methods directly and work with different interpolation methods. This common representation is given by a *rmregion*. In this section we show how to construct this common representation.

6.1. *A Common Representation for Interpolation Methods*

We construct a common representation for different interpolation methods, using *rmregions* as follows. Given two observations of the evolution of a region, we choose an interpolation method *H*, collect samples from the transformation created using *H*, with some granularity and approximate the trajectories of the vertices during interpolation using quadratic Bezier curves that pass through the sampled vertices, i.e. we construct the moving segments of the corresponding *rmregion*. The latter can only be performed if a correspondence between the vertices of the two geometries being interpolated is given or found. It makes sense to use the correspondence used by *H*, so we assume this correspondence is available. It is important to note that the set of samples collected always includes the first and the last shape created by *H*. Figure 9 shows snapshots of an interpolation and of an approximation of that interpolation created using the method described.

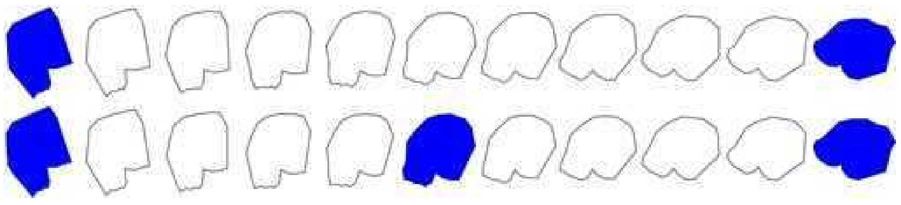

Figure 9. 11 snapshots of the evolution of a region created using the rigid interpolation method proposed in (Alexa *et al.* 2000, Baxter *et al.* 2008) (top) and our approximation strategy with 1 unit (3 samples) (bottom).

A quadratic Bezier curve has 3 control points and we have the option to choose if the curve passes through all its control points. Therefore, a *curve* type can be used to approximate the trajectory of a vertex between 3 consecutive samples. The *rmregion* and the *curve* types that define the trajectories of its vertices are defined in different times. Assuming that the sampling process uses a constant step (granularity), time can be transformed to curve time by normalizing it to the range $[0, 1]$. Different choices can be made w.r.t the granularity and the type of curve used to define the trajectories of the vertices. In this work we use quadratic Bezier curves whose control points are sampled vertices, a curve passes through all its control points, the granularity is defined by the number of samples to be collected, the samples are collected using a constant step and in a *urmregion* the trajectory of a vertex is defined using a single quadratic Bezier curve. If $2k+1$ samples are taken, $k \in \mathbb{Z}^+$, the corresponding *rmregion* will have k *urmregions*. This means that we need at least 3 samples to create an approximation of an interpolation, i.e. $k = 1$. The granularity is case dependent. The objective is to use the minimum number of samples while obtaining a good approximation w.r.t some criteria, e.g. the evolution of the area during interpolation. Studying the best strategy to choose the granularity is left for future work and investigation.

When using this procedure: (i) the rotation of objects with fixed shape is approximated not exact and (ii) a bad or unacceptable representation can be generated (this depends largely on the characteristics of the curves being approximated and the number of Bezier curves used in the approximation) e.g. if a large rotation exists and few samples are used. This procedure does not provide explicit mechanisms to detect, handle or avoid self-intersections during interpolation. In the next section, we define the operation *interpolate* to construct this common representation.

6.2. *Interpolate: region* $\times$ *instant* $\times$ *region* $\times$ *instant* $\times$ $\boldsymbol{H} \times \boldsymbol{P} \rightarrow$ *rmregion*

Given two observations of the evolution of a region o_σ and $o_{\sigma+1}$, taken at two instants of time, an interpolation method H, e.g. the rigid interpolation method proposed in (Alexa *et al.* 2000, Baxter *et al.* 2008), and extra parameters P that include (i) parameters specific to H, (ii) the correspondence between the vertices of o_σ and $o_{\sigma+1}$ used by H and (iii) the granularity of the approximation, *interpolate* creates a *rmregion* according to the procedure presented in Section 6.1 (See Figure 12 (C and D)).

Interpolate has two steps. First H is used to create an interpolation. Then, the interpolation is approximated to create a *rmregion*. The complexity of the first step depends on H. In the second step, in a straightforward implementation, we have to iterate through the k units of the *rmregion* that will be created and for each unit we have to iterate through the s segments of a sample. This is because we can get the position of the endpoints of a segment in the 3 samples in a unit in constant time. Therefore, the complexity in this case is $O(ks)$. In practice, k is expected to be a small number, so k is some constant. We note, however, that it is possible to iterate through the s segments of a sample only once and construct all the moving segments of the k units of the *rmregion* corresponding to the μ samples taken. Thus, the complexity of *interpolate* is $O(H^O + s)$, where H^O is the complexity of H in the first step and s is the number of segments in o_σ or $o_{\sigma+1}$ (this includes segments that are degenerated to a point).

7. Experimental Results

This section presents experimental results obtained when using the data model for *rmregions* and the algorithms for spatiotemporal operations proposed in this work. We start by emphasizing the following points. (i) We use real and synthetic data to perform the experiments presented in this section. (ii) The algorithms for spatiotemporal operations and the data model proposed in this work were implemented using Python 3 in a VMware virtual machine and are a non-optimized reference implementation. (iii) The region interpolation methods, librip and the polyhedra based model, and the spatiotemporal operations proposed in the moving objects databases literature used in this work to obtain results and perform experiments were used via the Secondo database (Güting *et al.* 2010) that was executed in a VMware virtual machine. We use librip as a reference to compare the approximations obtained when using the region interpolation methods proposed in the literature and our strategy. In the examples shown, librip provides the best representation that can be obtained using the region interpolation methods proposed in the literature. We use the polyhedra based model to compare execution time results, because it is more efficient when handling objects with a large number of units. (iv) We present and compare execution time results obtained using our strategy and the methods proposed in the literature, however, (*ii*) and (*iii*), above, should be considered when drawing conclusions from the results shown. The results obtained using our strategy do not include the execution time needed to create the interpolation that is approximated. This is because: this execution time depends on the interpolation method used, the original interpolation is supposed to be created only once and it seems to make sense to focus the analysis on the execution time of the strategy. (v) Few of the spatiotemporal operations proposed in the literature are actually implemented. This makes a more complete comparison between different methods difficult to perform and had an influence on the methods we chose to use. (vi) We found some problems when using some region interpolation methods implementation. Some of these problems had not been resolved in the version of Secondo that was used. (vii) The solver used to find the roots of the quartic polynomials used to find the intersections between a moving point and a moving segment and between two moving segments computes approximate solutions. This can cause numerical problems and make the use of optimizations that rely on exact solutions harder to use in practice. A video showing some experimental results is available at[1].

7.1. *Number of Units × Execution Time*

To compare the approximations created using the region interpolation methods proposed in the literature and quadratic Bezier curves we use two examples: a simple region, say sr, that rotates clockwise between two observations of its evolution (See Figure 12) and the iceberg presented in Section 3.3.

We consider the interpolation created using the rigid interpolation method proposed in (Alexa *et al.* 2000, Baxter *et al.* 2008) as our ground truth. We take different numbers of samples from the original interpolation using a constant step, create moving regions from those samples using quadratic Bezier curves and librip (these are moving regions with a different number of units) and compare the moving regions created, with the ground truth. We take 500 observations from the original interpolation (our ground truth) and 500 observations from the moving regions that were created. Then, we compare the representations obtained, visually, and we compare the evolution of the area and measure its maximum deviation w.r.t the ground truth (See Figures 10 and 11).

In figures 10 and 11, gt, ku, `soa_ku` and `bz_ku` stand for ground truth, moving region with k units and moving region with k units created using librip and Bezier

[1] `https://drive.google.com/file/d/1_C5yK5BjMpHOyRgO4scV62VwFvekDQDC/view?usp=sharing`

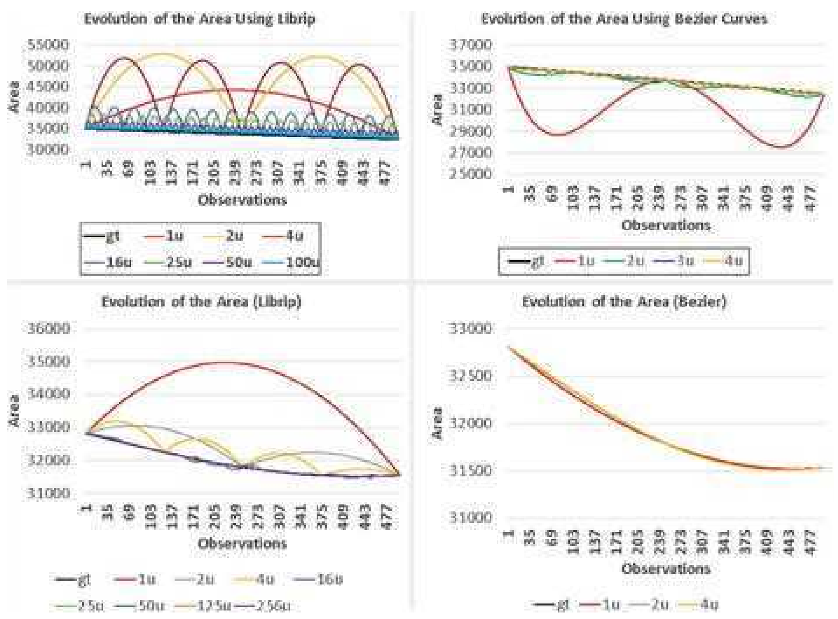

Figure 10. Evolution of the area of *sr* (top) and the iceberg (bottom) when using librip and Bezier curves.

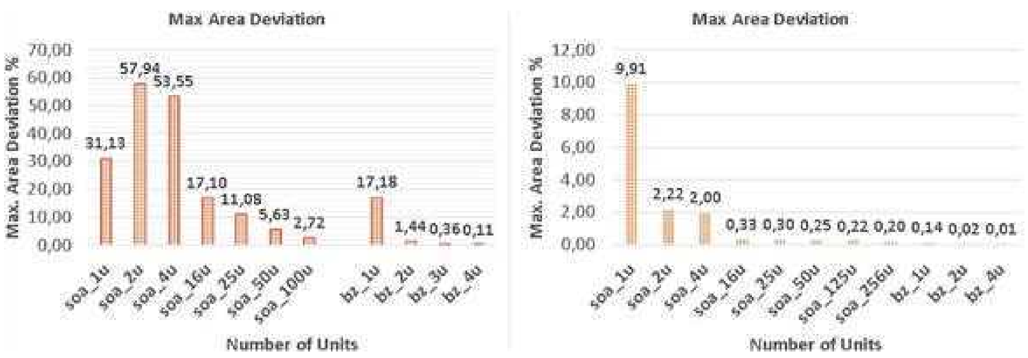

Figure 11. Maximum area deviation of the moving regions that were created w.r.t the ground truth for *sr* (left) and the iceberg (right).

curves, respectively.

When using librip we can observe the following. In the first case (Figure 10 (top left)) we can see a clear pattern in the evolution of the area of the moving region during its evolution. In this example, the moving region with 1 unit and the moving regions with $>$ 1 unit, if we consider that they all represent a rotation, represent the rotation of the object in opposite directions. This causes an unexpected result, i.e. the moving regions with 2 and 4 units have a bigger area deviation w.r.t the ground truth than the moving region with 1 unit (See Figure 12 (A and B)). There is also an effect that can be seen in the chart, but may not be completely obvious. As the number of units grows, each unit will be defined in a smaller interval of time. Because of the pattern of the evolution of the area, and depending on the amount of time in which a unit is defined, a wave effect will occur on the border of the object until a certain number of units is reached, in which case such effect cannot be perceived anymore. If the amount of time is small enough, this effect resembles a heartbeat. See a video demonstrating this effect here[2]. In the case of the iceberg, we can observe that even with $>$ 200 units

[2]https://drive.google.com/file/d/17oMPzE9eLpFPL9ezavk2_xST8d0qTjCi/view?usp=sharing

moving regions can have units where concavities simultaneously appear and disappear from/to a point. These cases corresponds to the *fluctuations* on the curves seen in Figure 10 (bottom left). The area of the object can shrink during a unit and grow during another unit, in the same moving region. If the intervals of time where the units are define correspond to relatively short periods of time these *fluctuations* will occur as shown in this video[3], where a moving region with 125 units is shown evolving during a short period of time. Because the rotation of the iceberg is relatively small, the deviation of the area is mainly caused by the representation of the concavities.

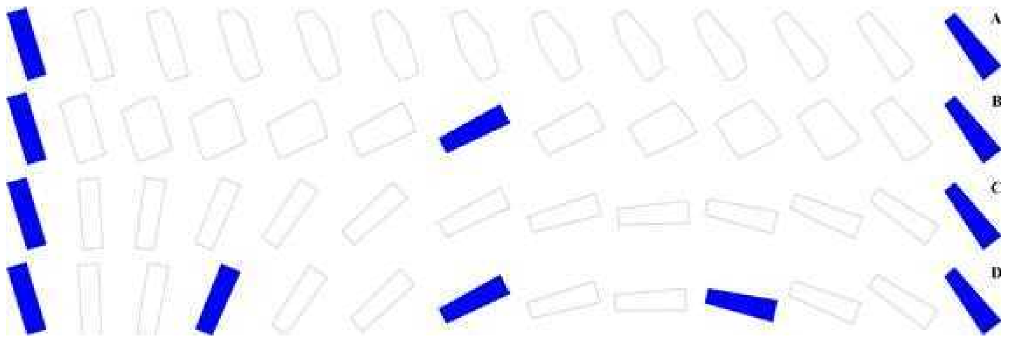

Figure 12. 13 snapshots of the evolution of sr created using: librip with 1 (A) and 2 units (B), the rigid interpolation method proposed in (Alexa *et al.* 2000, Baxter *et al.* 2008) (C) and quadratic Bezier curves with 2 units (D). In (A) and (C), the unit starts and ends at the first and last snapshot shown, respectively. In (B) and (D), the first and second units begin and end at the first and seventh and at the seventh and thirteenth snapshot shown, respectively. (D) was created using 5 samples collected from (C).

When using Bezier curves there is also a pattern. However, the pattern is much more subtle, because the moving regions approximate the ground truth very rapidly. Also, there are no *fluctuations* because a correspondence given by the user is used.

In the first case, we can observe that the moving regions created using Bezier curves approximate the ground truth very rapidly and with only 2 units the maximum area deviation is around 1.44%. In the case of the moving regions created using librip, a moving region with 100 units has a maximum area deviation of around 2.72%. It is important to note that the trajectories of the vertices during interpolation tend to approximate linear functions as the number of units grows (in which case there is no advantage in using quadratic bezier curves) (See Figure 11 (left)). We also have to take into consideration the wave effect that can occur and be more or less severe. Even if a reasonable number of units provides a reasonable maximum area deviation the corresponding representation can include this effect. If such effect is desirable, or to which extent it is acceptable, is, we believe, case and application dependent. In the second case, a moving region with 1 unit created using Bezier curves has a lower maximum area deviation w.r.t the ground truth than a moving region with 256 units created using librip and a moving region with 2 units created using Bezier curves has a maximum area deviation of 0.02% (See Figure 11 (right)).

Finally, we study the evolution of the execution time when querying when the iceberg and a moving point and the iceberg and another moving region intersect, when their evolution is created using quadratic Bezier curves (i.e. using our strategy) and the polyhedra based model. The iceberg is evolving between 2 observations and its evolution is represented using $1, 2, 4, 16, 25, 50, 125, 250$ and 256 units. The moving point has a linear trajectory and it is evolving during a unit. The second moving region is evolving between 2 observations (the first and last observations of mr_2, shown in

[3]`https://drive.google.com/file/d/1teCtDexbX4HcsqBJpjbtwk6uVMPxJ2q_/view?usp=sharing`

Section 7.2.2, Figure 14), during a unit. Table 1 presents the execution times obtained for the 2 cases.

Table 1. Execution times obtained when querying the time intervals when a moving region represented using a different number of units, created using quadratic Bezier curves and the polyhedra based model, intersects with a moving point with a linear trajectory (left) and with another moving region (right).

	Execution Time Msec		**Execution Time Msec**	
Num Units	**Bezier (AABB)**	**SOA**	**Bezier (AABB)**	**SOA**
1	4,00	27,25	331,00	538,00
2	9,00	34,25	604,00	690,80
4	15,00	49,75	981,00	1059,00
16	55,00	138,75	3797,00	3055,20
25	84,00	214,25	5322,00	4709,40
50	182,00	414,00	10606,00	9160,60
125	447,00	1067,00	26318,00	22837,20
250	857,00	2291,25	53282,00	47315,00
256	871,00	2381,75	54296,00	48107,20

In Table 1, Bezier (AABB) and SOA refer to execution time results obtained when querying when the moving objects intersect, when the evolution of the iceberg is represented using *k* units (*Num Units*) created using our strategy and the polyhedra based model, respectively. In the first case, the algorithms for spatiotemporal operations use an Axis-Aligned Bounding Box filter step to improve their performance. The execution times shown cannot be compared directly, because the polyhedra based model is running in a database, as mentioned in the beginning of Section 7. This introduces an overhead on the execution time that we cannot quantify precisely. The results can, however, be used as a reference. In the first case (Table 1 (left)), i.e. when querying the time intervals when the iceberg and a moving point intersect, we can observe that the execution times obtained using our strategy are reasonable. We note that in this case, when using 2 units we obtain a maximum area deviation of approx. 0.02%. This corresponds to an execution time of approx. 9 *msecs*. In the second case (Table 1 (right)), i.e. when querying the time intervals when the iceberg and another moving region intersect, we observe that the execution times obtained when using our strategy grow faster as the number of units grows. The execution times are still within reasonable bounds, when compared with the results obtained using the polyhedra based model. If using 2 units to estimate the evolution of the iceberg we obtain an execution time of approx. 604 *msecs*. Although we can expect that the execution time will grow faster when using our strategy as the number of units used to create the evolution of a *rmregion* grows, the key observation is that we need much less units to approximate the rotation of a region between 2 observations, with a very good quality. This suggest that our strategy is a reasonable alternative to the methods proposed in the moving objects databases literature in the cases discussed and presented. It is up to the user to choose which method to use to create the evolution of a phenomenon or part of its evolution.

7.2. *Spatiotemporal Operations Involving RMRegions*

This section presents experimental results obtained using a reference implementation of the algorithms for spatiotemporal operations proposed in Section 5.

7.2.1. *Moving Point × Rotating Moving Region*

We used 9 cases for testing, created using 3 rotating moving regions, mr_1, mr_2 and mr_3, with 41, 40 and 850 vertices, respectively, evolving between two observations, and 9 moving points, mp_1, ..., mp_9, with linear and quadratic trajectories. Figure 13 shows mr_1, mr_2 and mr_3. The execution time results presented here were obtained by executing each query 100 times for each case. *Intersects* uses a simple axis-aligned bounding boxes (AABBs) filter step to reduce the number of moving point *vs* moving segment intersection checks.

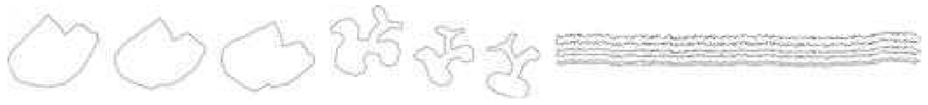

Figure 13. mr_1 (left), mr_2 (middle), mr_3 (right) and the samples used to generate their evolution.

Table 2 presents average execution time results for *intersects: mpoint × rmregion → mbool*, AVG ET[1], and *intersects: mpoint × rmregion → bool*, AVG ET[2], for the 9 cases considered. AVG ET, NIT, I, NV and MPTT stand for average execution time in seconds, the number of time intervals where the two moving objects intersect, if the two moving objects intersect, the number of vertices of the rotating moving region and the type of trajectory of the moving point (**L**inear or **Q**uadratic), respectively. In the case of the first operation: *touch*, *disjoint* and *inside* give similar results. Cases 8 and 9 have the largest average execution time: 0.026 sec and 0.016 sec, in the case of the first and the second operation, respectively. These are also the cases where the rotating moving region has the largest number of vertices. The number of vertices of the rotating moving region, the number of intersections (or not having intersections) and the type of trajectory of the moving point are important factors, but the number of moving point *vs* moving segment intersection checks performed will ultimately determine the execution time. Among other possible optimizations, a good (an efficient) filter step is needed to reduce the number of intersection checks.

Table 2. Execution times for *intersects: mpoint × rmregion → mbool* and *intersects: mpoint × rmregion → bool* for the 9 cases considered.

		mp × rmr → mbool		**mp × rmr → bool**			
Case	**M. Objects**	**AVG ET (sec)[1]**	**NIT**	**AVG ET (sec)[2]**	**I**	**NV**	**MPTT**
1	**mr1 x mp1**	0,00483	1	0,00209	True	41	L
2	**mr1 x mp2**	0,00460	2	0,00085	True	41	L
3	**mr1 x mp3**	0,00524	2	0,00088	True	41	Q
4	**mr2 x mp4**	0,00547	1	0,00258	True	40	Q
5	**mr2 x mp5**	0,00714	2	0,00170	True	40	Q
6	**mr2 x mp6**	0,00785	3	0,00105	True	40	Q
7	**mr2 x mp7**	0,00673	0	0,00664	False	40	Q
8	**mr3 x mp8**	0,02589	3	0,01549	True	850	Q
9	**mr3 x mp9**	0,02580	5	0,01541	True	850	Q

7.2.2. *Rotating Moving Region × Rotating Moving Region*

We used 9 cases for testing, created using 5 rotating moving regions, mr_1, ..., mr_5, with $41, 19, 40, 63$ and 850 segments, respectively, evolving between two observations, with different trajectories. Figure 14 shows the 5 rotating moving regions. *Intersects* uses a simple axis-aligned bounding boxes (AABBs) filter step to reduce the number of moving segment *vs* moving segment intersection checks. Table 3 presents average execution time results for *intersects: rmregion* $\times$ *rmregion* $\rightarrow$ *mbool*, AVG ET[1], and *intersects: rmregion* $\times$ *rmregion* $\rightarrow$ *bool*, AVG ET[2], for the 9 cases. *NV MR1* and *NV MR2* give the number of vertices of the two rotating moving regions involved, respectively. In the case of the first operation: *touch*, *disjoint*, *inside* and *overlaps* give similar results. Case 9 has clearly the largest average execution time, 6.642 sec. This is the case involving the two rotating moving regions with the largest number of vertices. In the case of the second operation, case 7 has the largest average execution time, 0.078 sec. This is the case where the two rotating moving regions do not intersect. Case 3 has the smallest average execution time by taking advantage of the fact that the two rotating moving regions are known to intersect at the end of their evolution. The results were obtained by executing each query 100 times for each case. Again, the main factor determining the execution time is the number of moving segment *vs* moving segment intersection checks.

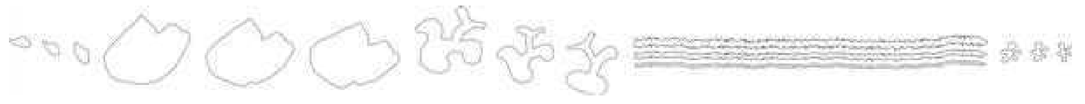

Figure 14. From left to right: mr_2, mr_1, mr_3, mr_5, mr_4 and the samples used to generate their evolution.

Table 3. Execution times for *intersects: rmregion* $\times$ *rmregion* $\rightarrow$ *mbool* and *intersects: rmregion* $\times$ *rmregion* $\rightarrow$ *bool* for the 9 cases considered.

		rmr × rmr → mbool		**rmr × rmr → bool**			
Case	**M. Objects**	**AVG ET (sec)[1]**	**NIT**	**AVG ET (sec)[2]**	**I**	**NV MR1**	**NV MR2**
1	**mr1 x mr2**	0,070	1	0,003	True	41	19
2	**mr1 x mr2**	0,050	1	0,017	True	41	19
3	**mr1 x mr2**	0,021	1	0,000	True	41	19
4	**mr3 x mr4**	0,248	1	0,016	True	40	63
5	**mr3 x mr4**	0,165	1	0,015	True	40	63
6	**mr3 x mr4**	0,151	1	0,048	True	40	63
7	**mr3 x mr4**	0,077	0	0,078	False	40	63
8	**mr3 x mr4**	0,095	2	0,057	True	40	63
9	**mr5 x mr5**	6,642	1	0,044	True	850	850

8. Conclusion

In this paper we propose a strategy to estimate and query the evolution of moving regions whose moving segments are allowed to rotate and change their length between two observations. This introduces a new class of moving regions called rotating moving regions. The strategy allows different interpolation methods to be used to create moving regions and is independent of the interpolation method used and its complexity. This is achieved by creating an approximation of the interpolation (created using some interpolation method) with a certain granularity using quadratic Bezier curves. This establishes a common structure for moving regions that is then used to represent and query their evolution allowing a more realistic representation of the continuous evolution of real-world phenomena in moving objects databases, in particular, in cases where a rotation occurs between two observations of the evolution of the phenomena. Another characteristic of this strategy is that moving points can have non-linear trajectories in a natural way. We present the implementation of a set of operations involving rotating moving regions and results obtained when using them. We study the use of a different number of units to represent the rotation of a moving region between two observations and the execution time of spatiotemporal operations using these representations. Experimental results show that our strategy is a reasonable alternative to the region interpolation methods proposed in the moving objects databases literature when a certain number of units has to be used to approximate the rotation of a region between two observations, with a certain quality, and can be used to complement these methods.

We do not consider the case when the trajectory of a moving segment and a moving point and the trajectory of two moving segments are collinear and studying the best strategy to choose where or when to collect samples during interpolation to obtain an approximation with a certain quality using the minimum number of samples, is left for future work and investigation. The strategy (i) can be applied to complex geometries, e.g. polygons with holes, (ii) can be used as an interpolation method on its own, (iii) segments can appear from and disappear to a point during interpolation, (iv) the rotation of objects with fixed shape can only be approximated, (v) a bad (or unacceptable) representation can be generated (this depends largely on the characteristics of the curves being approximated and the number of Bezier curves used in the approximation) e.g. if a large rotation exists and few samples are used, (vi) a correspondence between the vertices of the geometries being interpolated has to be given and (vii) the strategy does not provide explicit mechanisms to detect, handle or avoid self-intersections during interpolation. The primitives used to implement operations on *rmregion*s can be used as a base to, at least, detect if self-intersections occur during interpolation.

Data and Codes Availability Statement

The data and codes that support the findings of this study are available at `https://doi.org/10.6084/m9.figshare.17284877`.

Acknowledgement(s)

José Duarte has a research grant awarded by the Portuguese public agency for science, technology and innovation (FCT) under the reference PD/BD/142879/2018, financed by the Portuguese State Budget and co-financed by the European Union Budget through the European Social Fund (ESF) and the Centro Regional Operational Program (Centro 2020). This work is partially funded by FCT in the context of the projects UIDB/00127/2020, POCI-01-0145-FEDER-032636 and MIT-EXPL/ACC/0057/2021.

Disclosure statement

The authors report there are no competing interests to declare.

Notes on contributors

José Duarte is a Ph.D. student at the University of Aveiro with interests in spatiotemporal databases. He earned his MS and BS in Computer Science from the University of Aveiro.

Paulo Dias is an Assistant Professor at the Department of Electronics, Telecommunications and Informatics of the Universidade de Aveiro (Portugal) and a researcher at IEETA, a non-profit R&D institute affiliated with the same university. His main research interests are virtual and augmented reality, 3D reconstruction, computer vision, computer graphics, visualization, combination and fusion of data from multiple sensors.

José Moreira is an Assistant Professor at the Department of Electronics, Telecommunications and Informatics of the Universidade de Aveiro (Portugal) and a researcher at IEETA, a non-profit R&D institute affiliated with the same university. His main research interests are spatiotemporal databases, data provenance and time series.

References

Alexa, M., Cohen-Or, D., and Levin, D., 2000. As-rigid-as-possible shape interpolation. *In*: J.R. Brown and K. Akeley, eds. *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, 23–28 July 2000, USA. ACM Press/Addison-Wesley Publishing Co., 157–164.

Baxter, W., Barla, P., and Anjyo, K., 2008. Rigid shape interpolation using normal equations. *In*: K. Anjyo and F.X. Sillion, eds. *Proceedings of the 6th International Symposium on Non-Photorealistic Animation and Rendering*, NPAR '08, 9–11 June 2008, New York, NY, USA. ACM, 59–64.

Forlizzi, L., *et al.*, 2000. A data model and data structures for moving objects databases. *SIGMOD Record*, 29 (2), 319–330.

Güting, R.H., Behr, T., and Düntgen, C., 2010. SECONDO: A platform for moving objects database research and for publishing and integrating research implementations. *IEEE Data Engineering Bulletin*, 33 (2), 56–63.

Güting, R.H., *et al.*, 2000. A foundation for representing and querying moving objects. *ACM Transactions on Database Systems*, 25 (1), 1–42.

Güting, R.H. and Ding, Z., 2004. A simple but effective improvement to the plumb-line algorithm. *Information Processing Letters*, 91 (6), 251–257.

Güting, R.H. and Schneider, M., 2005. *Moving objects databases.* 1st ed. The Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann.

Heinz, F. and Güting, R.H., 2016. Robust high-quality interpolation of regions to moving regions. *GeoInformatica*, 20 (3), 385–413.

Heinz, F. and Güting, R.H., 2018. A data model for moving regions of fixed shape in databases. *International Journal of Geographical Information Science*, 32 (9), 1737–1769.

Heinz, F. and Güting, R.H., 2020. A polyhedra-based model for moving regions in databases. *International Journal of Geographical Information Science*, 34 (1), 41–73.

Liu, Y., Yan, H., and Martin, R., 2011. As-rigid-as-possible surface morphing. *Journal of Computer Science and Technology*, 26 (3), 548–557.

McKenney, M. and Frye, R., 2015. Generating moving regions from snapshots of complex regions. *ACM Transactions on Spatial Algorithms and Systems*, 1 (1), 4:1–4:30.

McKenney, M., Viswanadham, S.C., and Littman, E., 2014. The CMR model of moving regions. *In*: C. Zhang, A. Basalamah, A.M. Hendawi and P. Nguyen, eds. *Proceedings of the 5th ACM SIGSPATIAL International Workshop on GeoStreaming*, IWGS '14, 4 November 2014, New York, NY, USA. ACM, 62–71.

McKenney, M. and Webb, J., 2010. Extracting moving regions from spatial data. *In*: D. Agrawal, P. Zhang, A.E. Abbadi and M.F. Mokbel, eds. *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '10, 3–5 November 2010, New York, NY, USA. ACM, 438–441.

Moreira, J., Dias, P., and Amaral, P., 2016. Representation of continuously changing data over time and space: Modeling the shape of spatiotemporal phenomena. *In*: *2016 IEEE 12th International Conference on e-Science (e-Science)*, 23–27 October 2016. IEEE Computer Society, 111–119.

Tøssebro, E. and Güting, R.H., 2001. Creating representations for continuously moving regions from observations. *In*: C.S. Jensen, M. Schneider, B. Seeger and V.J. Tsotras, eds. *Advances in Spatial and Temporal Databases*, 12–15 July 2001, Berlin, Heidelberg. Springer Berlin Heidelberg, 321–344.

Yan, H., Hu, S., and Martin, R., 2004. Morphing based on strain field interpolation. *Computer Animation and Virtual Worlds*, 15 (3-4), 443–452.