A framework for the management of deformable moving objects

José Duarte
DETI / Universidade de Aveiro
Campus Universitário de Santiago
3810-193 Aveiro
hfduarte@ua.pt

Paulo Dias
DETI / Universidade de Aveiro
Campus Universitário de Santiago
3810-193 Aveiro
paulo.dias@ua.pt

José Moreira
DETI / Universidade de Aveiro
Campus Universitário de Santiago
3810-193 Aveiro
jose.moreira@ua.pt

ABSTRACT

There are a growing number of applications and services based on spatiotemporal data in the most diverse areas of knowledge and human activity. This comes from the emergence of technologies that can obtain data about real-world phenomena over time. These data consist of discrete observations of the evolution i.e., the changes in position or shape and extent, of these phenomena over time but we are often interested in the representation of its continuous evolution. As a consequence, we need to have methods capable of representing the evolution of a phenomenon at all times between two known observations.

The representation of the continuous evolution of moving regions, i.e., entities (or objects) whose position, shape and extent change continuously over time, is an ongoing research topic. In this paper we present a framework for the representation and management of moving objects data (in particular moving regions data) that uses compatible triangulation and rigid interpolation methods to represent their continuous evolution over time. The aim is to obtain a realistic representation of such evolution and at the same time maintain compatibility with existing spatial data management systems currently in use. We also present a spatiotemporal database extension that uses this framework and usage examples showing how to perform operations combining numeric, spatial, and spatiotemporal data.

CCS Concepts

• **Information systems → Information systems applications → Spatial-temporal systems**.

• **Information systems → Data management systems → Database design and models → Data model extensions**.

Keywords

Moving Objects, Spatiotemporal Data Management, Morphing.

SAC'18, April 9-13, 2018, Pau, France.

http://dx.doi.org/xx.xxxx/xxxxxxx.xxxxxxx

1. INTRODUCTION

Currently, there are many tools and technologies able to monitor and record the evolution of real-world phenomena over time, such as, satellite or aerial images tracking the movement of icebergs at the Antarctic, the propagation of forest fires or coastal erosion, or video microscopy and fluorescence microscopy recording the size and shape of cells over time. However, the modelling, management and processing of spatiotemporal data are complex. Despite recent advances, the support provided by existing solutions is insufficient and do not cover the whole spectrum of potential applications. It is often required to implement time-consuming and complex programs tailored to solve a specific problem from a particular domain, which cannot be easily applied to problems from the same or other domains. In addition, these solutions often have limited functionality because development is difficult and costly. Thus, the study of solutions to represent, manage and process large spatiotemporal datasets has the potential to enable easier implementation of those programs and boost the development of applications in several domains, e.g., environmental and climate sciences, agriculture and biomedicine.

The databases community is working on data models and query languages for the management of spatiotemporal data, including the representation of moving objects, i.e., objects whose position or shape change continuously over time. Their main focus is on efficient data management and on topological issues concerning the representation of spatiotemporal data in databases. However, the methods used do not always yield a representation of spatiotemporal phenomena that is close to their evolution in reality. On the other hand, there are numerous works on morphing techniques to represent the transformation of planar shapes, e.g., polygons and polylines, over time. The focus is on obtaining continuous representations of the spatial transformations that are smooth and realistic at all times. Thus, although it is not a trivial task, the use of morphing techniques in databases has the potential to enable obtaining representations of spatiotemporal phenomena that are more realistic than those proposed in the databases literature.

This work focuses on the management of spatiotemporal data using continuous models of time and space. The aim is to bring morphing techniques used successfully in animation packages to the field of spatiotemporal data management and thus improve their ability to represent moving objects in information systems. This paper presents the design and architecture of a spatiotemporal data management framework. The data model and operations that it implements follow the guidelines proposed in [6,

9, 21] and the novelty is the use of the concept of *mesh* to represent the evolution of moving regions over time. For that purpose, we use the compatible triangulation method presented in [8] and the interpolation method presented in [1, 3].

This framework has two components. A library that is independent of any client using it and that provides a framework to represent, analyze and manipulate moving objects, in particular moving regions, and a spatiotemporal database extension for PostgreSQL and PostGIS that uses this library to manipulate and analyze moving objects. This extension allows moving objects to be stored in PostgreSQL and analyzed and manipulated in a convenient way using the Structured Query Language (SQL).

The results present usage examples to show that this framework: allows the analysis and manipulation of moving objects and works with real data; can be used to store moving objects in a database management system and to analyze and manipulate these data using SQL; and can obtain a realistic representation of the continuous evolution of moving regions over time.

The paper is organized as follows. Section 2 presents an overview on spatiotemporal data models and query languages. Section 3 presents the concepts on continuous and discrete models proposed on spatiotemporal databases literature that are used in this work. It also presents the compatible triangulation and the interpolation methods that are used in this work. Section 4 presents the data model, the operations and the architecture of a framework for dealing with moving objects, in particular moving regions, which uses triangulation-based morphing techniques. Section 5 presents a spatiotemporal database extension that uses the proposed framework. Section 6 presents the results we obtained when using synthetic and real data. Section 7 presents the conclusions and guidelines for future work.

2. RELATED WORK

The representation modes to deal with spatiotemporal data are raster (discrete) and vector (continuous), and the choice depends on the applications' requirements [14]. Discrete modes use data models based on the discretization of time, space or both dimensions. The general approach is to assign timestamps to a variety of formats such as feature classes (geometries) [7, 11, 13], and mosaic data [4, 18, 26, 28]. The timestamps can be time instants or time intervals. Although several examples exist in the literature, it is worth highlighting the data model and query language based on the point set theory proposed in [27]. Discrete representations are easier to implement but the accuracy, storage requirements and performance are strongly influenced by the temporal and spatial resolutions. Conversely, continuous representations allow for more compact data structures and enable modeling definable entities or events, e. g., the size and shape of a cell or the frontline of a forest fire, which are implicit in discrete models.

The most well-known data model and query language for representing and querying moving objects uses Abstract Data Types (ADTs) [5, 9]. The main reason for its success is that ADTs may be smoothly built into extensible Database Management Systems (DBMSs), such as the Secondo prototype [10], and object-relational DBMSs [15, 21, 22, 29]. Moving objects, e.g., moving points, moving lines and moving regions, are represented using the *sliced representation* proposed in [6], i.e., they are represented as an ordered collection of units and each unit represents the evolution (i.e., the changes in position or shape and extent) of the moving object between two consecutive observations. Therefore, methods are required to model the spatial behavior of moving objects between observations. These methods should obtain representations that are close to the objects' actual shapes and movements at all times. Additionally, storage requirements and computation time should be low to enable the processing of large datasets.

The first algorithm proposed in the literature to create the so-called moving regions from observations in spatiotemporal databases is presented in [25]. This algorithm is used and extended in recent works [12, 16, 17]. The basic principle is to use moving segments to interpolate the evolution of a moving region during a unit. The focus is on creating representations that are topologically valid at all times, while keeping compatibility with the underlying spatial database systems. However, the methods using this algorithm may cause deformation of the geometries estimated during the interpolation and the approximation errors may be too big to be neglected in scientific work, namely: if a rotation exists, the geometries tend to inflate at the middle point of the interpolation; the methods used to deal with concavities either do not perform well with noisy data [12, 20, 25] or make the shapes approximately convex during interpolation, thus causing deformation [16, 17].

In this work we propose using morphing techniques to represent the evolution of moving regions. The transformation of planar shapes is a well-known problem and there are already morphing algorithms that can be used to estimate the shape of a moving object at any point in time. In particular, we propose the use of methods based on triangulation because they allow the representation of complex shapes as a set of triangles and thus, to make the implementation of complex spatiotemporal operations, e.g., the intersection between objects, easier. Additionally, there are morphing techniques based on planar triangulation that use algebraic methods with well-defined computational costs.

3. BACKGROUND

3.1 Spatiotemporal Data Models

The framework proposed in this work uses the data models presented in [6, 9] as a reference. In [9] the authors present an abstract data model to represent and query moving objects and in [6] the authors present a discrete data model to implement the model proposed in [9]. This discrete model proposes the following type constructors: the base types *int*, *real*, *string* and *bool*, the spatial types *point*, *points*, *line* and *region*, the temporal types *instant* and *range*, the unit types *ureal*, *upoint*, *upoints*, *uline* and *uregion* and the *mapping* type. Type *instant* represents a point in time, *range* represents sets of pairwise disjoint intervals, the unit types represent the continuous evolution of an entity (or object) during an interval, e.g., the changes of position, shape and size of an iceberg, and *mapping* assembles sets of units. It also proposes the concept of *sliced representation* used to represent the moving types. According to the authors, the *sliced representation* (Figure 1) decomposes the development (or evolution) of a value into fragments called slices (or units) and the evolution within a unit is described by a function. Generically, a unit is a pair (I, $v(t)$) where I is the time interval where the unit is defined and $v(t)$ is a function (or a representation of a function) that gives the position, shape and size of an entity (or object) during I. The begin and end

instants of I represent two consecutive observations of the entity (or object) being represented.

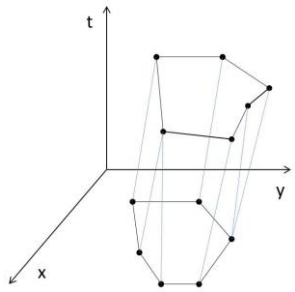

Figure 1: Sliced representation.

3.2 Morphing

The representation of the evolution of a moving region during a unit is given by a function that should preserve the physical characteristics of the object being represented and generate valid geometries at all times. That is, we are interested in an as realistic as possible representation of the continuous evolution of moving regions over time. In [2] the author studied the use of compatible triangulation [8] and rigid interpolation [1, 3] methods to represent the evolution of moving regions over time, i.e., to solve the Region Interpolation Problem. The use of these methods implies that a region is represented by a mesh (Figure 2, right) and they are used to obtain a realistic representation of the continuous evolution of moving regions over time.

There are two main steps to calculate the evolution of a moving region during a unit. The first is to calculate a compatible triangulation between the polygons representing the shape of the moving region at the beginning and ending of the unit's time interval. The method used in this work, proposed in [8], receives 2 polygons, a source, P, and a target, Q, polygons and triangulates them generating two meshes, P* and Q*. P and Q must have a one-to-one correspondence between their boundary points. Note that, there are several methods to make the number of vertices coincide without changing the border of the polygons [19]. P* and Q* have a one-to-one correspondence between their triangles (Figure 2).

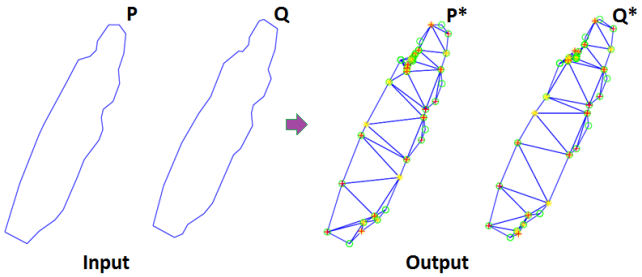

Figure 2: Compatible triangulation method input (left) and output (right).

The second step consists of using a rigid interpolation method to compute the transformation of the moving region during a unit. In this work, we use the method proposed in [1, 3], where the affine transformation of a triangle is decomposed into a product of three matrices obtained by applying the Single Value Decomposition (SVD). Since there is an affine matrix for each triangle, shared vertices will have more than one position during a transformation. To ensure that shared vertices have a single position during transformations [1] proposes a solution based on the least squares method, while [3] proposes a solution based on normal equations. Indeed, the two methods are equivalent, but the latter presents an elegant formulation that allows estimating the position of the vertices using only products of matrices. The formula is $V(t) = -H^{-1}.G(t)$, where V(t) is a vector denoting the unknown positions of the vertices of a mesh at time t, and $-H$ and G are matrices calculated using SVD and normal equations. The matrices $-H$ and $G(t)$ are calculated using three matrices: $\mathbb{P}$, $R\alpha$ and $R\gamma(t)$. The matrices $\mathbb{P}$ and $R\alpha$ are independent of time (t) and give the translation, skewing and scaling of the triangles, and $R\gamma(t)$ gives the rotation component. The calculation method of these matrices is out of the scope of this paper and can be found in [3]. Briefly, the rigid interpolation method receives two meshes, a source, P*, and a target, Q*, with a one-to-one correspondence between their triangles and computes the interpolation components used to interpolate the mesh between P* and Q*. Each triangle of P* will be rotated, scaled, translated and transformed to its corresponding triangle in Q*.

4. A FRAMEWORK FOR MOVING OBJECTS

We implemented a framework for moving objects, in particular, for moving regions that is independent from any client or application using it, called SPTMesh. SPTMesh is a C++ library. It provides a C++ and a C APIs and its main goals are to enable the analysis and manipulation of moving objects. To implement the compatible triangulation and rigid interpolation methods, SPTMesh introduces a new spatial type called *mesh* and uses the architecture of the GEOS[1] C++ library as a reference.

The rigid interpolation method, that SPTMesh implements, uses linear algebra operations to compute the Moore-Penrose pseudo-inverse of a matrix and to perform the SVD of a matrix. There are several libraries for linear algebra available, such as: Armadillo [24], Eigen[2], Eigen do Better[3], LAPACK[4] and OpenBLAS[5]. In particular, Eigen, Eigen do Better and Armadillo are template-based C++ libraries. We considered using Eigen, but Eigen does not provide a suitable SVD operation. Armadillo meets all our needs for linear algebra operations.

The set of data types implemented by SPTMesh has as its base the types proposed in [6] with some changes. The spatial types implemented in SPTMesh (except for the *mesh* type) are compliant with the Open Geospatial Consortium (OGC) standards for spatial objects. It introduces the following new types: the spatial type *mesh* and the types *umesh*, *function* and *mmesh*. Type *mesh* is a triangulated polygon used to represent a moving region at a given instant, *umesh* represents the evolution of a *mesh* during a time interval (between two consecutive observations), *function* represents a function in the mathematical sense (e.g., a linear or a quadratic function) and *mmesh* represents the evolution of a *mesh* over time. It defines the types *interval* and *period* instead of the type *range* and the MOVING types instead of the type *mapping*.

[1] http://geos.osgeo.org/

[2] http://eigen.tuxfamily.org/

[3] http://eigendobetter.com/

[4] http://www.netlib.org/lapack/

[5] http://www.openblas.net/

SPTMesh does not consider lines, collections and regions with holes and implements the data types presented in Table 2. It uses the GEOS C++ library for manipulation and analysis of 2D geometries in the Cartesian plane. Table 1 shows the external dependencies of SPTMesh.

Table 1: SPTMesh external dependencies

Dependency	Usage
Armadillo	Linear algebra operations.
BLAS, LAPACK	Armadillo dependencies to provide various matrix decompositions, e.g., SVD.
GEOS	C++ library for manipulation and analysis of 2D geometries in the Cartesian plane and for implementation of spatial operations.

4.1 Data Structures

Table 2 presents the SPTMesh data structures used to implement the data types that it considers. Figure 3 shows the SPTMesh type system.

Table 2: SPTMesh data structures

Data Type(s)	Data Structure Implementation
BASE	Types *int*, *real* and *bool* are implemented using the C++ int, double and bool data types.
instant	Type *instant* is implemented using the C++ long data type (we are not using dates with locales and time zones, yet).
SPATIAL	Type *mesh* is implemented in the class Mesh. The other SPATIAL types point, linestring, polygon, multipoint, multilinestring, multipolygon and geometrycollection are provided by the GEOS C++ library.
interval and *period*	Type *interval* represents an interval of time in the form [$instant_i$, $instant_j$[, *period* represents sets of intervals. They are implemented in the template classes Interval and Period, respectively. Interval supports the 4 types of intervals but we only use closed-open intervals.
function	Type *function* represents functions in the mathematical sense. It only considers the constant function, f = C, the linear function, f = b*x* + C and the quadratic function, f = ax^2 + b*x* + C. It is implemented in the UnitFunction class.
UNIT	UNIT types represent the evolution of moving objects (points and regions) and the changes of bool (true or false) and real values during a time interval. They are implemented in the classes: UnitBool, UnitReal, UnitPoint and UnitMesh.
MOVING	MOVING types represent moving objects (*mpoint* and *mmesh*) and values that change continuously (*mreal*) and in discrete steps (*mbool*) over time. They are implemented in the classes: MovingBool, MovingReal, MovingPoint and MovingMesh.

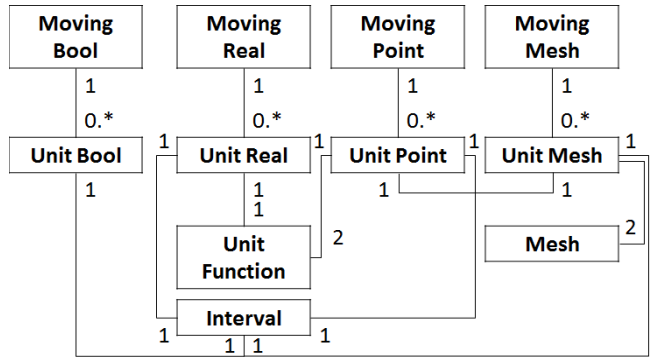

Figure 3: The SPTMesh type system.

Types *mesh* and *umesh* are the most important data types in SPTMesh.

A *mesh* has a boundary, a set of interior points, i.e., the Steiner points added by the compatible triangulation method and a set of non-overlapping triangles. The Mesh class uses two vectors to hold the mesh's boundary vertices (a set of points) and Steiner points, respectively, and a vector to hold the indices of the points of the mesh's triangles. These indices point to boundary and Steiner points.

Type *umesh* is a set of tuples $\{(I, t_b, t_e, c, P^*, Q^*, \mathbb{P}, R\alpha, R\gamma(t)) \mid I \in \text{Interval}, t_b, t_e \in \text{long long}, c \in \text{UnitPoint}, P^*, Q^* \in \text{Mesh}, \mathbb{P}, R\alpha \in \text{mat}, R\gamma(t) \in \text{vector<double>}\}$, such that:

- I is the time interval in which the unit is defined.
- t_b and t_e represent the begin and end instants of the original interval, respectively. They are an implementation detail that we are not going to discuss in this paper.
- c is a UnitPoint that represents the evolution of the position of the mesh's centroid during I.
- P* and Q* represent the original source and target meshes at t_b and t_e, respectively, and P* and Q* have a one-to-one correspondence between their triangles.
- mat is an Armadillo data type that represents a matrix.
- vector is the C++ standard library vector data type.
- $\mathbb{P}$, $R\alpha$ and $R\gamma(t)$ are the interpolation components used to interpolate the mesh during I (see Section 3.2).

4.2 Operations on Moving Types

SPTMesh implements only a subset of the spatiotemporal operations proposed in [9] (Table 3). We chose to implement a set of operations according to the following criteria: implement atomic operations, i.e., operations that can be used as building blocks to implement other operations, use all the implemented moving types and implement operations from the different classes of operations proposed in the literature.

Table 3: Operations on MOVING types defined in SPTMesh

Class of Operation	Operation
Predicates	equals, intersects
Set Operations	intersection

Numeric	area
Projection to Domain and Range	deftime
Interaction with Domain and Range	atinstant, atperiod, present
Constructors	unit, moving

4.3 Continuity for the Unit Types

Two units can meet at an instant. To avoid situations like instantaneous appearance and disappearance or instantaneous growing and shrinking we need to establish continuity for the UNIT types. We establish continuity for UnitPoint and UnitMesh as follows:

- Two UnitPoint objects are continuous if the distance between their positions at the time when they meet is less than or equal to a ξ, i.e., distance(p_1, p_2) $\leq \xi$. SPTMesh defines ξ = 0.00001. This value was not strictly established and needs to be validated.
- Two UnitMesh objects are continuous if distance(mi, mj) $\leq \xi p \wedge \frac{m_i \cap m_j}{m_i \cup m_j} \leq \delta s$, mi, mj $\in$ Mesh. That is, if the distance between their centroids is less or equal to a constant value and the ratio between the intersection and the reunion of their areas is less or equal to some other constant value. SPTMesh defines ξp = 0.5 and δs = 0.95. These values were empirically set during the tests phase using real data. However, given the limited size of our dataset the constants and formulas used to establish continuity need further evaluation and should be fine-tuned when using larger datasets. Note that these constants may also be application-dependent.

We do not establish continuity for UnitBool and UnitReal for the following reasons: UnitBool represents a value that changes in discrete steps over time, UnitReal can represent the evolution of quantities with different natures, e.g., the evolution of the area and the evolution of the distance between two moving regions. Continuity for UnitReal might be established in the future.

4.4 Architecture

The SPTMesh architecture has as a reference the GEOS C++ library architecture (Figure 4**Error! Reference source not found.**). SPTMesh provides a C API on top of a C++ implementation and that should be the interface used by applications linking to it so that these applications still work without relinking when the framework is upgraded.

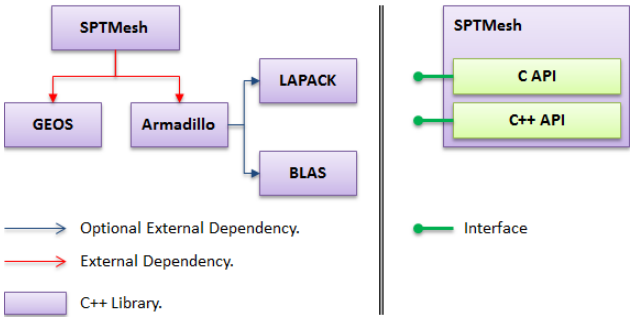

Figure 4: SPTMesh architecture and available APIs.

4.5 Spatiotemporal Well-Known Text Form for the UNIT and MOVING Types

SPTMesh provides a Spatiotemporal Well-Known Text (STWKT) form for the UNIT and MOVING types (Table 5 and Table 6). The STWKT provides a convenient way of expressing unit and moving objects and it is particularly useful to create moving objects and to update them with new evolutions. The STWKT form was inspired by the OGC standard Well-Known Text (WKT) form for spatial objects.

We use the notation as given in Table 4.

Table 4: Notation

Symbol	Description
t_b, t_e $\in$ *instant*	Begin and end instants of an *interval*.
$v \in$ *bool*	A value in {true, false} or equivalently in {1, 0}.
v_b, v_e $\in$ *double*	Begin and end values, respectively.
type, typeX, typeY $\in$ *int*	The type of the interpolation function. typeX and typeY represent the type of interpolation function for the x and y components, respectively. An interpolation function can be of the following types: a constant function, a linear function and a quadratic function. An interpolation function is represented by the *function* data type
x_b, y_b, x_e, y_e $\in$ *double*	Coordinates of a moving point at the begin and end instants of the units' interval.
x_i y_i $\in$ *double*	The x and y coordinates of a point (vertex).

Table 5: STWKT form for the UNIT types

Data Type	STWKT Form
ubool	UNITBOOL(t_b t_e v)
ureal	UNITREAL(t_b t_e v_b v_e type)
upoint	UNITPOINT(t_b t_e x_b y_b x_e y_e typeX typeY)
umesh	UNITMESH(t_b t_e, (x_1 y_1, …, x_n y_n), (x'_1 y'_1, …, x'_n y'_n))

Table 6: STWKT form for the MOVING types

Data Type	STWKT Form
mbool	MOVINGBOOL(UNITBOOL$_1$, …, UNITBOOL$_n$)
mreal	MOVINGREAL(UNITREAL$_1$, …, UNITREAL$_n$)
mpoint	MOVINGPOINT(UNITPOINT$_1$, …, UNITPOINT$_n$)
mmesh	MOVINGMESH(UNITMESH$_1$, …, UNITMESH$_n$)

5. A SPATIOTEMPORAL DATABASE EXTENSION FOR POSTGRESQL

We implemented a spatiotemporal database extension for PostgreSQL called MeshGIS. MeshGIS is a C library that uses SPTMesh to analyze and manipulate moving objects. MeshGIS allows the moving objects provided by SPTMesh to be stored in a PostgreSQL database and manipulated using SQL. We also provide SQL data types and operations that bind to the MeshGIS types and operations.

5.1 Data Structures

The PostgreSQL backend is implemented in C. It might, however, be possible to load functions written in other languages, namely: C++, FORTRAN or Pascal, into PostgreSQL and we can write PostgreSQL extensions using C++, if certain guidelines [6] are followed. PostgreSQL also supports procedural languages[7], e.g., PL/Python that allows PostgreSQL to use functions written using Python, but they have several significant limitations[8]. Following the examples of PostGIS[9] and Hermes[10] we decided to implement MeshGIS as a C library.

The MeshGIS data structures to represent SPTMesh types are defined as C typedef structs and they are presented in Table 7.

Table 7: MeshGIS data structures to represent SPTMesh types

MeshGIS Data Structure	Description
ArrayOfX	A generic array to hold units or other elements, e.g., UnitReal, UnitBool, UnitPoint and Matrix2x2.
UnitFunction	A SPTMesh UnitFunction.
UnitInterval	A SPTMesh Interval.
UnitBool	A SPTMesh UnitBool.
UnitReal	A SPTMesh UnitReal.
UnitPoint	A SPTMesh UnitPoint.
UnitMesh	A SPTMesh UnitMesh.
SerializedPostgreSQLObject	This data structure is used to send and retrieve moving objects to/from PostgreSQL for storage, analysis and manipulation. It has the same structure as the GSERIALIZED data structure that is a PostgreSQL data type for variable size user defined data types and provides a serialized form used primarily by PostGIS.
SerializedMovingObject	An abstract data type that represents any type of moving object.
SerializedMovingX	Represents the SPTMesh MovingBool, MovingReal and MovingPoint objects.
SerializedMovingMesh	A SPTMesh MovingMesh.

MeshGIS also defines data structures to represent the interpolation components (Table 8).

Table 8: MeshGIS data structures to represent the interpolation components

Data Structure	Description
Matrix2x2	A 2x2 matrix representation. Used to hold the scale matrix of a triangle.
Matrix2x3	A 2x3 matrix representation. Used to hold the coordinates' transformation matrix of a triangle.
Triangles	Used to hold the vertices' ids of a triangle.

In this version MeshGIS defines specific data structures to represent the interpolation components. This is to be changed in the future so that MeshGIS is decoupled from interpolation method details.

MeshGIS defines operations that use all the operations implemented in SPTMesh for the moving types, namely constructors, operations to add and delete units from moving objects, operations to test and obtain the intersection of 2 moving regions at an instant and operations to get a moving region at an instant or period.

We provide SQL operations and types that bind to the operations and types provided by MeshGIS. The SQL types provided are MovingBool, MovingReal, MovingPoint and MovingMesh.

5.2 Architecture

The MeshGIS architecture has as a reference the PostGIS architecture (Figure 5). MeshGIS uses SPTMesh through its C API.

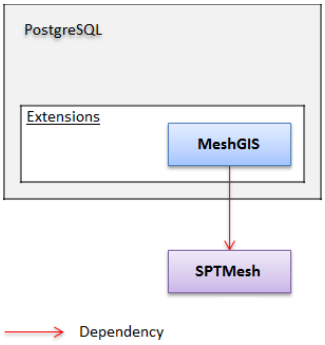

Figure 5: MeshGIS architecture.

MeshGIS has 2 main use cases: 1) store a moving object in a PostgreSQL database and 2) analyze or manipulate a moving object stored in a PostgreSQL database.

In 1) a client can send a STWKT form of a moving object to MeshGIS using SQL. MeshGIS will use the SPTMesh C API to create a valid moving object from its STWKT form. Then that object will be converted to the corresponding MeshGIS object, serialized to a SerializedPostgreSQLObject and sent to PostgreSQL for storage.

In 2) MeshGIS receives the moving object from PostgreSQL in a serialized form, i.e., in a SerializedPostgreSQLObject. This object will be converted to the corresponding MeshGIS object. Then MeshGIS will use SPTMesh to create the moving object and to analyze and manipulate it as needed.

6. RESULTS

This section presents application examples to demonstrate the use of SPTMesh and MeshGIS. Two datasets are used for validation purposes:

[6] https://www.postgresql.org/docs/9.4/static/xfunc-c.html

[7] PostgreSQL documentation: Chapter 39 - Procedural Languages

[8] PostgreSQL documentation: Chapter 35 - Extending SQL

[9] http://postgis.net/

[10] http://infolab.cs.unipi.gr/?page_id=1999

- Synthetic data, i.e., data created manually to test specific conditions that do not occur when using real data.
- A set of real data extracted from a sequence of satellite images tracking the movement of 2 icebergs in the Antarctic [23] using the methods implemented in [19].

6.1 Tests Using Synthetic Data

We used synthetic data to test specific conditions that we consider relevant, like situations where there are rotations ≥ 2π (Figure 6) and the 180º rotation of an object (Figure 7).

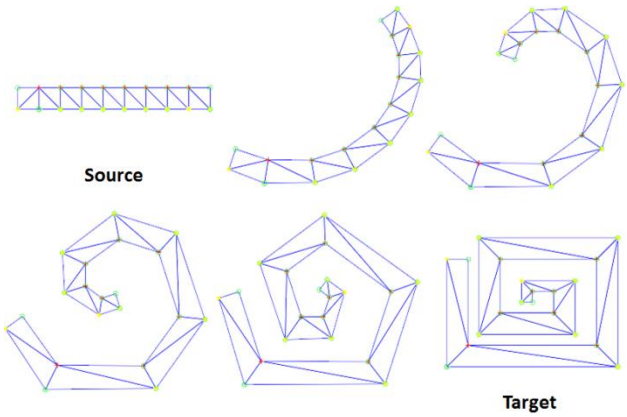

Figure 6: Coil interpolation test.

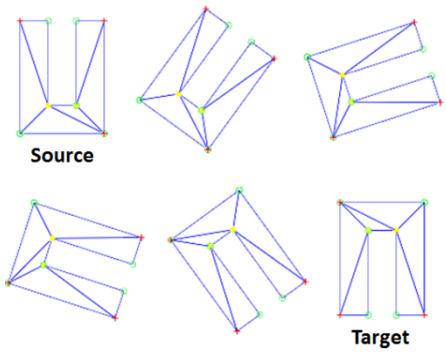

Figure 7: 180º rotation test.

This example shows that the solution adopted in this work is able to deal with complex cases involving the rotation of objects and concavities, two issues that can be found when using the methods proposed in [12, 16, 17, 25].

6.2 Tests using Real Data

To demonstrate MeshGIS we start by creating a table in PostgreSQL to store the spatiotemporal data about the evolution of the 2 icebergs. For this we use the MovingMesh data type. Note that the values of the time intervals used for testing purposes are numbers and do not represent valid dates.

```
CREATE TABLE db.icebergs
(
    id      integer,
    name varchar(50),
    mobj movingmesh
)
```

We can insert data into the icebergs table using the following functions:

INSERT INTO db.icebergs(id, name, mobj) VALUES(1, ‘ice 1’, *ST_MovingMesh_FromSTWKT*('MOVINGMESH((1000 2000, (1052 987, ..., 1034 941), (1055 999, ..., 1001 875)))'));

INSERT INTO db.icebergs(id, name, mobj) VALUES(2, ‘ice 2’, *ST_MovingMesh_CreateEmpty*());

The first command creates a moving object called 'ice 1' with a unit describing its evolution during the time interval [1000, 2000[. The two sequences of coordinates given in the MovingMesh STWKT represent the position and the shape of the iceberg at instants 1000 and 2000, respectively. Internally, MeshGIS uses SPTMesh to construct the moving object from its STWKT form, i.e., SPTMesh will apply the compatible triangulation method and compute the interpolation components for this unit. The second command creates a moving object with an empty spatiotemporal component.

The command below displays the contents of the table after the execution of the previous commands (Table 9).

SELECT * FROM db.icebergs;

Table 9: Results of the select command in the icebergs table

id	name	mobj
1	ice 1	MOVINGMESH((1000 2000, (1052 987, 1090 1037, …, 1034 941), (1055 999, …, 1001 875)))
2	ice 2	MOVINGMESH EMPTY

Note that the representation of the moving values uses the STWKT form presented in Table 6.

We can add data to a record in the icebergs table independently of the method used to create it. For example, we can add a unit, i.e., a new evolution of the iceberg during a time interval, to the record with id = 1.

UPDATE db.icebergs SET mobj = *ST_Add_UnitMesh*((SELECT mobj FROM db.icebergs WHERE id = 1), ‘UNITMESH(2000 3000, (1001 875, …, 979 848), (1030 942, …, 996 896))', false) WHERE id = 1;

After inserting the icebergs’ data in the icebergs table we can ask questions about their properties and the relationships that they establish with each other.

We can obtain information about the icebergs at a specific period (Figure 8) or instant (Figure 9).

SELECT *ST_Get_AtPeriod*(mobj, 'PERIOD(1100 4500)') FROM db.icebergs WHERE id = 2;

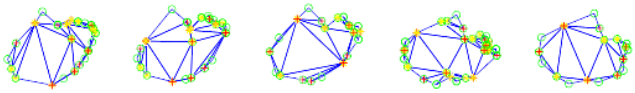

Figure 8: Iceberg 2 at instants 1100, 2000, 3000, 4000 and 4500.

SELECT *ST_Get_AtInstant*(mobj, 1500) FROM db.icebergs WHERE id = 1;

Figure 9: Iceberg 1 at instant 1500.

The result of the query depicted in Figure 8 is also a moving region and so, it represents the shape of the object at all times between 1100 and 4500. The figure displays only 5 snapshots for demonstration purposes. The result of the query depicted in Figure 9 is a projection of an umesh at time instant 1500, which SPTMesh converts automatically into a SPATIAL type that can be used, for example, as the input of a PostGIS operation.

We can obtain the intersection of the two icebergs at a specified instant (Figure 10). As in the real dataset the two icebergs do not intersect, we created two moving meshes based on the real dataset that intersect each other at time instant 1000.

SELECT *ST_Intersection*((SELECT mobj FROM db.icebergs WHERE id = 3), (SELECT mobj FROM db.icebergs WHERE id = 4), 1000);

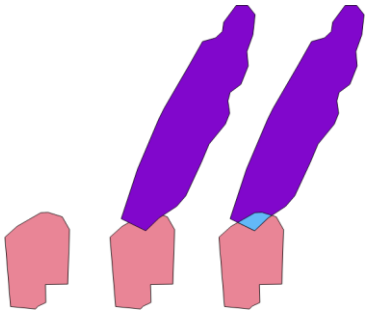

Figure 10: Intersection of 2 icebergs at instant 1000 in light blue.

We can use PostGIS to get the area of the intersection between the 2 moving objects at instant 1000 (Table 10). That is, we can use PostGIS functions to process results obtained by MeshGIS.
SELECT ST_Area(ST_GeomFromText(*ST_Intersection*((SELECT mobj FROM db.icebergs WHERE id = 3), (SELECT mobj FROM db.icebergs WHERE id = 4), 1000)));

Table 10: Using PostGIS to get the area of the intersection of 2 moving regions at instant 1000

Intersection Area (in abstract units)
1815.203

Finally, we can use the operations provided by MeshGIS to see the evolution of a moving object during a time interval, e.g., during a unit (Figure 11). We can choose the interpolation step, i.e., the frequency used to interpolate the evolution of the moving object.

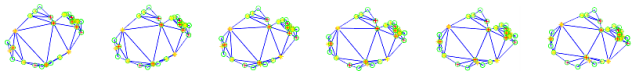

Figure 11: Iceberg 2 interpolation during an interval of time.

7. CONCLUSIONS

We implemented a framework for moving objects, in particular moving regions, called SPTMesh in a library that is independent from any application using it. A region is represented by a mesh, i.e., a triangulated polygon, and the framework uses compatible triangulation and rigid interpolation methods to obtain a realistic representation of the continuous evolution, i.e., the continuous changes of the position, shape and extent, of moving regions over time. Our framework for moving objects does not consider lines, collections and regions with holes and implements only a subset of the spatiotemporal operations proposed in the literature.

We implemented a spatiotemporal database extension for PostgreSQL called MeshGIS as a C library that uses SPTMesh to analyze and manipulate moving objects and we provide Structured Query Language (SQL) types and operations that bind to the MeshGIS types and operations. MeshGIS makes it possible to store moving objects in PostgreSQL and to analyze and manipulate them in a convenient way using SQL. It is also possible to use PostGIS to further process MeshGIS spatial results obtained from operations on moving types. We also provide a convenient representation of the moving types called Spatiotemporal Well-Known Text (STWKT).

Tests show that SPTMesh is application-independent and can be used, in particular, with spatiotemporal database extensions. The tests that were performed obtained interesting and promising results. We intend to continue the development of SPTMesh and MeshGIS in the future. We want to test SPTMesh with larger and diverse datasets and extend it so that it can represent regions with holes and collections. We also want to implement a larger set of spatiotemporal operations, in particular the projection of moving regions into the plane and the intersection of moving regions over time.

8. REFERENCES

[1] Alexa, M. et al. 2000. As-rigid-as-possible shape interpolation. Proceedings of the 27th annual conference on Computer graphics and interactive techniques - SIGGRAPH ’00 (2000), 157–164.

[2] Amaral, A. 2015. Representation of spatio-temporal data using Compatible Triangulation and Morphing techniques. Aveiro University.

[3] Baxter, W. et al. 2008. Rigid shape interpolation using normal equations. NPAR ’08 Proceedings of the 6th international symposium on Non-photorealistic animation and rendering (2008), 59–64.

[4] Bothwell, J. and Yuan, M. 2010. Apply concepts of fluid kinematics to represent continuous space–time fields in temporal GIS. Annals of GIS. 16, 1 (Apr. 2010), 27–41. DOI:https://doi.org/10.1080/19475681003700872.

[5] Cotelo Lema, J. et al. 2003. Algorithms for Moving Objects Databases. The Computer Journal. 46, 6 (2003), 680–712.

[6] Forlizzi, L. et al. 2000. A Data Model and Data Structures for Moving Objects Databases. Proceedings of the 2000

ACM SIGMOD International Conference on Management of Data (2000), 319–330.

[7] Gebbert, S. and Pebesma, E. 2014. A temporal GIS for field based environmental modeling. Environmental Modelling and Software. 53, (2014), 1–12. DOI:https://doi.org/10.1016/j.envsoft.2013.11.001.

[8] Gotsman, C. and Surazhsky, V. 2004. High quality compatible triangulations.

[9] Güting, R.H. et al. 2000. A Foundation for Representing and Querying Moving Objects. ACM Trans. Database Systems. 25, 1 (2000), 1–42. DOI:https://doi.org/10.1145/352958.352963.

[10] Güting, R.H. et al. 2010. SECONDO : A Platform for Moving Objects Database Research and for Publishing and Integrating Research Implementations. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering. 33, 2 (2010), 56–63.

[11] Hayashi, H. et al. 2016. Composition of simulation data for large-scale disaster estimation. Proceedings of the Second ACM SIGSPATIALInternational Workshop on the Use of GIS in Emergency Management - EM-GIS ’16 (New York, New York, USA, 2016), 1–8.

[12] Heinz, F. and Güting, R.H. 2016. Robust high-quality interpolation of regions to moving regions. GeoInformatica. 20, 3 (2016), 385–413. DOI:https://doi.org/10.1007/s10707-015-0240-z.

[13] Jaziri, W. et al. 2015. Using Temporal Versioning and Integrity Constraints for Updating Geographic Databases and Maintaining Their Consistency. Journal of Database Management. 26, 1 (Jan. 2015), 30–59. DOI:https://doi.org/10.4018/JDM.2015010102.

[14] Koubarakis, M. 2003. Spatio-temporal databases : the CHOROCHRONOS approach. Springer.

[15] Matos, L. et al. 2012. Representation and Management of Spatiotemporal Data in Object-relational Databases. Proceedings of the 27th Annual ACM Symposium on Applied Computing (New York, NY, USA, 2012), 13–20.

[16] Mckenney, M. and Webb, J. 2010. Extracting Moving Regions from Spatial Data. Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems (San Jose, California, 2010), 438–441.

[17] Mckennney, M. and Frye, R. 2015. Generating Moving Regions from Snapshots of Complex Regions. ACM Trans. Spatial Algorithms Systems. 1, 1 (2015), 1–30. DOI:https://doi.org/10.1145/2774220.

[18] Merino, L. et al. 2012. An Unmanned Aircraft System for Automatic Forest Fire Monitoring and Measurement. Journal of Intelligent & Robotic Systems. 65, 1-4 (Jan. 2012), 533–548. DOI:https://doi.org/10.1007/s10846-011-9560-x.

[19] Mesquita, P. 2013. Morphing Techniques For Representation of Geographical Moving Objects. Universidade de Aveiro.

[20] Moreira, J. et al. 2016. Representation of continuously changing data over time and space: Modeling the shape of spatiotemporal phenomena. 2016 IEEE 12th International Conference on e-Science (e-Science) (Oct. 2016), 111–119.

[21] Pelekis, N. et al. 2006. Hermes - A Framework for Location-Based Data Managment. EDBT’06 Proceedings of the 10th international conference on Advances in Database Technology (Munich, Germany, 2006), 1130–1134.

[22] Pelekis, N. et al. 2014. On the Support of Mobility in ORDBMS. International Journal of Knowledge-Based Organizations. 4, 1 (Jan. 2014), 38–64. DOI:https://doi.org/10.4018/ijkbo.2014010103.

[23] RossSea Subsets: 2004. http://rapidfire.sci.gsfc.nasa.gov/imagery/subsets/?project=antarctica&subset=RossSea&date=11/15/20. Accessed: 2016-09-20.

[24] Sanderson, C. and Curtin, R. 2016. Armadillo: a template-based C ++ library for linear algebra. The Journal of Open Source Software. 1, (2016), 26.

[25] Tøssebro, E. and Güting, R. 2001. Creating Representations for Continuously Moving Regions from Observations. Proceedings of the 7th International Symposium on Advances in Spatial and Temporal Databases (2001), 321–344.

[26] Villarroya, S. et al. 2016. SODA: A framework for spatial observation data analysis. Distributed and Parallel Databases. 34, 1 (Mar. 2016), 65–99. DOI:https://doi.org/10.1007/s10619-014-7165-7.

[27] Viqueira, J.R. and Lorentzos, N.A. 2007. SQL extension for spatio-temporal data. The VLDB Journal. 16, 2 (2007), 179–200. DOI:https://doi.org/10.1007/s00778-005-0161-9.

[28] Vizcaino, P. and Pistocchi, A. 2014. Use of a Simple GIS-Based Model in Mapping the Atmospheric Concentration of γ-HCH in Europe. Atmosphere. 5, 4 (Oct. 2014), 720–736. DOI:https://doi.org/10.3390/atmos5040720.

[29] Zhao, L. et al. 2011. STOC: Extending Oracle to Support Spatiotemporal Data Management. Springer, Berlin, Heidelberg. 393–397.