In Proceedings of the 5th ACM SIGSPATIAL International Workshop on GeoStreaming (IWGS '14).

The CMR Model of Moving Regions

Mark McKenney, Sarita C. Viswanadham, & Elizabeth Littman
Department of Computer Science
Southern Illinois University Edwardsville
{marmcke,sviswan,elittma}@siue.edu

ABSTRACT

Many natural phenomena can be nicely represented by concepts of moving regions. For example, hurricanes, rain clouds, pollution zones, etc., change shape and position over time. Current models of moving regions have proven to be difficult to translate effectively to implementation for two reasons: i) algorithms for operations, such as intersection, are difficult to implement, and ii) creating instances of moving regions from data sources is difficult. In this paper, we create a new model of moving regions at the abstract, discrete, and implementation levels that overcome the difficulties of previous models. The CMR model focuses moving region representation at temporal instants, allowing the model to focus on temporally static regions; thus, algorithms for data collection and operations can be implemented using largely 2-dimensional algorithms, which are well understood. The result is a model that aligns well with data collection techniques, can be implemented easily, and allows complex movement patterns to be easily depicted.

1. INTRODUCTION

Moving object databases focus on the storage and analysis of spatial objects that change in shape and position over time. Traditional moving object types include moving points, moving lines, and moving regions. For example, a vehicle can be represented as a moving point, a fast moving river that changes course over time may be represented as a moving line, and a raincloud that moves across a state can be represented as a moving region. Moving object data exist in many fields, including meteorology, climatology, biology, forestry, etc.

Much research in the literature investigates moving object data models, algebras, and analysis techniques; however, few prototype systems exist, and systems that implement moving regions are especially rare. The lack of moving region systems, in particular, seems to stem from two characteristics of existing moving region models i) generating moving region data from sensors is challenging, and ii) the algorithms proposed to implement operations on moving regions are complex and operate in 3-dimensional space. Progress has been made on the first characteristic in the form of algorithms that can build moving regions from a series of snapshots of static regions; however, algorithms on moving regions remain quite difficult to implement.

In this paper, we propose a new model of moving regions called the Component based Moving Region (CMR) model. The CMR model aims to achieve a model of moving regions that meets the following goals:

1. The model aligns with data collection mechanisms such that moving region data can be easily integrated into CMR systems from sensors.

2. Users can create moving regions with complex movement patterns easily.

3. Algorithms to compute operations, such as geometric set operations, use mostly 2-dimensional algorithms that are well understood and are currently used in non-moving spatial systems.

4. The model is compatible with existing moving object models such that the algebras developed for existing models apply to the CMR model as well.

The first goal is achieved by focusing the model around *known* data. For example, moving region data is often collected from sensors; these sensors, be they image based, video, aggregates of temperature sensors, etc., effectively provide timestamped snapshots of moving regions. Thus, to create a temporally continuous moving region from these snapshots, one must interpolate between them. When an interpolation occurs, a guess is being made as to how the region behaves in between the snapshots. Therefore, the CMR model is designed to store the snapshots explicitly, and use a generic motion function to generate the interpolation when needed. This design aligns directly with data collection techniques, allows users to control the detail of data by inserting as many snapshots as they wish, and allows domain specific motion functions to be used in particular applications.

The second goal of the CMR model is achieved by separating the semantic interpretation of a region from its physical representation. For example, users are able to create a region containing what is usually considered to be invalid structure, e.g., faces that overlap, and holes that do not intersect faces. Allowing such structure provides flexibility in creating moving regions, and flexibility in creating motion patterns. When the region is required for analysis, a valid region is extracted from the seemingly invalid structure using region type verification algorithms.

Finally, the third goal of the algorithm is achieved by, again, focusing on snapshots of a moving region at particular time instants. Most of geometric operations will occur over temporally static regions at fixed time instants, and the motion between the instants

ACM SIGSPATIAL GIS '14 Dallas, Texas USA

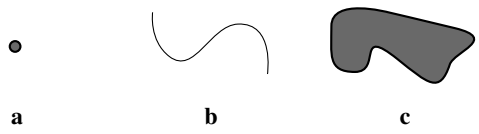

Figure 1: Example of a simple point (a), line (b) and region (c).

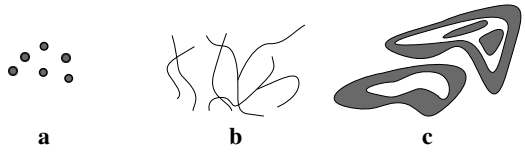

Figure 2: Example of a complex point (a), line (b) and region (c).

will again be interpolated. Taking this approach allows traditional 2-dimensional geometric algorithms to be used compute snapshots such that the regions in between the snapshots can again be interpolated.

Section 2 reviews related work. We begin introducing the CMR model at the abstract level in Section 3. In order to define the model correctly, we present definitions of traditional spatial types and build the model from them. Methods to discretize the abstract model are discussed in Section 4. In Section 5, we provide algorithms to compute operations over CMR model moving regions. An example in Section 6 demonstrates the use of the model. Finally, in Section 7, we draw some conclusions.

2. RELATED WORK

Databases that support the storage, management, and querying of spatiotemporal data have seen many milestones [9]. Among the various models, moving object models have been of particular interest. Moving object models align well with the representation of many natural phenomena, and have grown out of concepts of space that are used in geographic information systems, making them a natural choice for extensions of space into the temporal realm. In Section 2.1, we review the traditional spatial types. In Section 2.2, we review models of moving objects.

2.1 Spatial Objects

Spatial data modeling began with the development of the *simple spatial objects*: simple points, simple lines, and simple regions (Figure 1). Simple points represent a single point in space, simple lines are connected structures, and simple regions contain a single *face* and no *holes* in the face. The simple spatial types suffered from inability to represent many aspects of geographic reality, and could not ensure type closure under operations.

Complex spatial object models emerged as solution to the problems encountered by the simple object models. The complex spatial types consist of complex points, complex lines, and complex regions (Figure 2). A complex point contains many individual points, a complex line represents possibly disconnected networks, and a complex region can contain multiple faces, each containing zero or more holes. Formal definitions for complex types, based on point set theory, are provided in [10]. In section 3, we require definitions of some of the simple and complex types; however, we use a different formalization based on the spatial type definitions presented in [4, 7] that is somewhat more concise.

2.2 Moving Objects

Moving object data models have been proposed in the literature at both the abstract and discrete levels [5, 6, 11, 13, 1]. Most notably, an algebra describing moving object types of moving point, moving line and moving region, together with operations over the types, is developed in [5, 1]. The general approach in that model is extend the traditional complex types into the temporal domain by providing a function τ that maps instants in a set *time* to an arbitrary type α:

$$\tau(\alpha) : time \rightarrow \alpha$$

The application of the τ function to the complex types results in the corresponding moving object types:

$$\tau(point) \rightarrow moving\ point$$
$$\tau(line) \rightarrow moving\ line$$
$$\tau(region) \rightarrow moving\ region$$

Note that these definitions themselves are quite unrestrictive, allowing for object movement that may not be expected, or even considered to be possible. For example, a region can simply spring into existence for an instant, then disappear only to re-emerge at a different location in the next instant. In observable reality, there are few objects that can instantaneously translate themselves over space in a time instant in such a disconnected way. Thus, it is sometimes useful to impose continuity requirements on moving objects, requiring them to exist for a time interval, no matter how short, after appearing.

In our model, we define moving objects at the abstract level in much the same way as discussed above, with one major difference. The difference is that the model discussed above maps time instants to valid, temporally static types in order to produce a valid moving object. For example, a valid region is one in which its faces are disjoint or meet at a discrete number of points; thus a valid moving region is created from valid complex regions, ensuring that the complex region extracted from a moving region at any time instant is valid. In our model, we purposefully define a type of static region that contains structural properties that violate the traditional type definition of complex regions. For example, we allow faces of a region to intersect, and holes to exist outside of a region face; however, we also define a semantic interpretation of that structure that results in a valid region once the structure is interpreted. In this way, we provide type consistency and achieve flexibility in defining moving objects, representing motion, and capturing moving objects from sensor sources. However, it should be noted that because our region model is always interpreted as a valid region, it remains consistent with the algebra defined in [5]. In fact, the purpose for allowing structural abnormalities (as opposed to the traditional type definitions) is to create simple algorithms for data analysis over moving regions.

The discrete model of moving objects presented in [1] defines techniques for discretizing the abstract model. In short, moving objects are represented as a collection of *interval* objects in which discretized boundaries of the objects are defined as they move over a time interval; the name for this representation is the *slice* representation. Discretized object boundaries are considered to be polygonal line strings. Each interval effectively defines the path of each end point of a segment forming the boundary of a spatial object as it travels across the interval. As such, each end point of a line segment can be assigned its own unique movement function; this approach differs from our approach in which a global motion function is used to generate the motion of objects across intervals. In addition to discrete data models, the slice representation also provides algorithms for some spatial operations over moving objects. In particular, the algorithm for computing the intersection of two

moving regions is shown; this algorithm is a 3-dimensional algorithm in the sense that it operates over 3-dimensional objects. We provide a moving region intersection algorithm for the CMR model that uses a combination of 3-dimensional triangle/triangle intersection algorithms and well known 2-dimensional spatial algorithms in order to generate a moving region corresponding to the intersection of two input moving regions. The design of the CMR model to take advantage of more simple algorithms for implementing operations is deliberate, and is a strength of the model.

Finally, the models described in [5, 1] are, at least in part, implemented in the prototype system in [2]. However, it is not clear if the full algebras are implemented, and in particular, if the moving region operations were successfully implemented. In fact, other than that system, there is a distinct lack of moving region prototype systems. This stems from the difficulty of creating moving region data that fits the slice representation of moving regions, and the complexity of the algorithms needed to perform analysis on them. The CMR model, on the other hand, is designed to align with data collection mechanisms, and to reduce algorithms over moving region types to collections of algorithms over temporally static types when possible.

3. ABSTRACT DATA MODEL

In this section, we provide definitions of temporally static regions, and then provide a mathematical model of moving regions that will be later transformed to a discrete representation. Our approach to defining the spatial types is to create mappings from points in the Euclidean plane to *labels*. Points in the plane that are mapped to the same label are then considered as a point set. If these point sets exhibit certain properties, then they will form valid regions. Before we proceed, some notational definitions are necessary. Our definitions are based upon formal models of spatial partitions in [4, 7].

Let (X,T) be a topological space[1] with topology $T \subseteq 2^X$, and let $S \subseteq X$. The *interior* of S, denoted by S°, is defined as the union of all open sets that are contained in S. The *closure* of S, denoted by $\overline{S}$ is defined as the intersection of all closed sets that contain S. The *exterior* of S is given by $S^- := (X-S)^\circ$, and the *boundary* or *frontier* of S is defined as $\partial S := \overline{S} \cap \overline{X-S}$. An open set is *regular* if $A = \overline{A^\circ}$. The type of regular open sets is closed under intersection. In this paper, we deal with topological space $\mathbb{R}^2$.

Every member of a regular open point set in the plane is contained in a *neighborhood* of points that are also members of that set. We define the neighborhood of a given point $p = (p_x, p_y)$ in two dimensions as the set of points forming a circle around p with an infinitesimally small radius r. We define the neighborhood operation which returns the set of points contained in the neighborhood of p:

$$N := \mathbb{R}^2 \rightarrow 2^{\mathbb{R}^2}$$
$$N(p) = \{(x,y) | (x-p_x)^2 + (y-p_y)^2 = (r)^2\}$$

We define regions as a special case of spatial partitions [4]. In [4], the authors define the concept of a spatial mapping as a mapping from points in $\mathbb{R}^2$ to labels. Points with identical labels are denoted "blocks". Spatial partitions are then defined as spatial mappings such that blocks representing regions are regular open point sets, and blocks representing boundaries of regions are labeled with the labels of all adjacent regions. We are concerned with a special case of spatial partitions that represents regions as defined in [10]. A *spatial mapping* is a mapping from points in the topological plane to members of the power set 2^A where A is the set of labels that are used in a specific partition , i.e., $\pi : \mathbb{R}^2 \rightarrow 2^A$.

Definition 1 A *spatial mapping* is a total mapping $\pi : \mathbb{R}^2 \rightarrow A \bigcup 2^A$.

The application of a function $f : A \rightarrow B$ to a set of values $S \subseteq A$ is defined as $f(S) := \{f(x) | x \in S\} \subseteq B$. The inverse function $f^{-1} : B \rightarrow 2^A$ of f is defined $f^{-1}(y) := \{x \in S | f(x) = y\}$. f^{-1} is a total function and f^{-1} applied to a set yields a set of sets.

In this paper, we are concerned with defining spatial structures composed of the traditional spatial data types of point, line, and region; therefore, we are concerned with spatial mappings that map points in Euclidean space to labels in the set $A = \{\bullet, \perp\}$. This leads to the possibility of up to three blocks resulting from a spatial mapping: i) the block with label $\{\bullet\}$, used to denote the interior of a region, ii) the block with label $\{\bullet, \perp\}$, used to indicate line or point structures that have no area, and iii) the block with label $\{\perp\}$, used to indicate the exterior of spatial objects. Note that we provide alternate notations for interior, exterior, and boundary blocks for spatial mappings (as opposed to the traditional notation for spatial objects) since the the interiors, exteriors, and boundaries of some spatial objects are defined differently (notably lines). Spatial objects are formed from spatial mappings with restrictions; we use the usual notation for interior, exterior, and boundary on those objects.

Definition 2 *The* structures *of a* spatial mapping *are:*

(i)	$\rho(\pi) := \bigcup_{r \in \pi^{-1}(\{\bullet\})} r$	$(interior)$
(ii)	$\beta(\pi) := \bigcup_{r \in \pi^{-1}(\{\bullet,\perp\})} r$	$(boundary)$
(iii)	$\varepsilon(\pi) := \bigcup_{r \in \pi^{-1}(\{\perp\})} r$	$(exterior)$

where $\rho(\pi), \beta(\pi)$ *and* $\varepsilon(\pi)$ *are each point sets.*

A spatial region is then defined as a spatial mapping where the interior and exterior are regular open point sets, and the boundary is the intersection of the closure of the interior and exterior. We also provide the standard notation for interior, exterior and boundary of a region

Definition 3 A *region* is a spatial mapping $\pi_r : \mathbb{R}^2 \rightarrow 2^A$ where*:*

(i)	$\forall r \in \{\rho(\pi_r), \varepsilon(\pi_r)\}$ *:r* is a regular open set
(ii)	$\beta(\pi_r) = \bigcup_{s=\overline{r}-r \mid r \in \{\rho(\pi_r), \varepsilon(\pi_r)\}} s$
(iii)	$\pi_r^\circ = \rho(\pi_r)$
(iv)	$\partial\pi_r = \beta(\pi_r)$
(v)	$\pi_r^- = \varepsilon(\pi_r)$

A *simple region* is a special case of region that consists of one, connected face with no holes.

Definition 4 A *simple region* is a region $\pi_s : \mathbb{R}^2 \rightarrow 2^A$ where:

(i)	$\rho(\pi_s)$ is a connected regular open point set
(ii)	$\varepsilon(\pi_s)$ is a connected regular open point set
(iii)	$\pi_s^\circ = \rho(\pi_s)$
(iv)	$\partial\pi_s = \beta(\pi_s)$
(v)	$\pi_s^- = \varepsilon(\pi_s)$

A *simple point* consists of a single (x,y) coordinate in euclidean space. Points have the unique feature that their interiors are empty, therefore, a *simple point* is a spatial mapping with the restrictions that the cardinality of the boundary point set will be one, and the interior point set will be equal to the empty set:

[1] In a topological space, the following three axioms hold [3]: (i) $U,V \in T \rightarrow U \cap V \in T$, (ii) $S \subseteq T \rightarrow \bigcup_{U \in S} U \in T$, and (iii) $X \in T, \emptyset \in T$. The elements of T are called *open sets*, their complements in X are called *closed sets*, and the elements of X are called *points*.

Definition 5 *A simple point* is a spatial mapping $\pi_p : \mathbb{R}^2 \rightarrow 2^A$ *where:*

(i)	$\rho(\pi_p) = \emptyset$
(ii)	$\lvert\beta(\pi_p)\rvert = 1$
(iii)	$\overline{\varepsilon(\pi_p)} = \mathbb{R}^2$
(iv)	$\pi_s^\circ = \rho(\pi_s)$
(v)	$\partial\pi_s = \beta(\pi_s)$
(vi)	$\pi_s^- = \varepsilon(\pi_s)$

For the purposes of this paper, we require line structures that do not contain loops, and that are connected. Note that we do allow a line to branch. We denote this type of line a *simple line*. Eliminating loops is specified by requiring the exterior point set of a line to be connected. From the perspective of a spatial mapping, lines are similar to points in that every point in the point set describing the closure of a line will carry both the interior and exterior labels. The boundary of a line is considered to be the end points of the line, i.e., a line is defined as a one-dimension open point set embedded in two-dimensions. Thus, the boundary of the line is the closure of the line minus the original line, just as with two-dimensional open point sets. It is convenient to define the end points as points on the line with a neighborhood of cardinality 1. Note that due to the one dimensional nature of lines, the β function returns a closed point set. Because a line is a one dimensional structure and the exterior is a two-dimensional open set, the closure of a line's exterior will be equivalent to $\mathbb{R}^2$; this constraint enforces the one dimensional structure of a line:

Definition 6 *A simple line* is a spatial mapping $\pi_l : \mathbb{R}^2 \rightarrow 2^A$ *where:*

(i)	$\rho(\pi_l) = \emptyset$
(ii)	$\beta(\pi_l) \neq \emptyset$ and is a one-dimensional, connected, regular closed point set
(iii)	$\overline{\varepsilon(\pi_l)} = \mathbb{R}^2$
(iv)	$\pi_l^\circ = \{p \in \beta(\pi_l) \lvert N(p) > 1\}$
(v)	$\partial\pi_l = \{p \in \beta(\pi_l) \lvert N(p) = 1\}$
(vi)	$\pi_l^- = \varepsilon(\pi_l)$

We now define a *structural region* as the basis for our data model of moving regions. The structural region definition departs from traditional spatial type definitions in that the structure of the region does not explicitly imply the interpretation of the type; for example, traditional regions are defined such that the structure of a valid region type is a valid region. In other words, no special interpretation of the structure is necessary. Structural regions differ from this approach in that a structural region contains components that, without interpretation, define an invalid region; however, structural regions must be interpreted using an *extraction function* to extract the valid region defined by the structural region instance. This becomes clear from the fact that a structural region is defined as four sets: a set of faces, a set of holes, a set of lines, and a set of points. The set of faces consists of simple regions, and imposes no restrictions on the regions. simple regions in the face set are allowed to overlap, share boundaries, etc. Similarly, the set of holes is a set of simple regions that have no restrictions other than they must adhere to the simple region definition (e.g., they must be simple cycles); holes are not required to lie in a face, for instance. The set of points contains simple points, and the set of lines contains lines as defined previously. The point and line set are unnecessary for representing regions, but are required for the transition to moving regions. We use the term *component* of a structural region to refer to a member of the set of faces, holes, lines, or points.

The interior of a structural region is defined as the union of all faces minus the union of all holes; thus, holes that do not fall in a face have no effect (again, this property becomes important in the transition to moving regions). Because faces are defined as regular open point sets, the boundary of the regions is computed from the closure of the interior. Finally, the exterior is the difference of the embedding space and the interior. Thus, the interior, boundary, and exterior are interpreted from the structural components defining the region. Let $[D]$ indicate the set of all valid instances of type D:

Figure 3: A structural region (a) and the interpretation of its interior, exterior, and boundary (b).

Definition 7 *A* structural region *S is a tuple* $S = (F, H, L, P)$ *where:*

(i)	$F := \{f \in [\pi_s]\}$
(ii)	$H := \{h \in [\pi_s]\}$
(vi)	$L := \{l \in [\pi_l]\}$
(vii)	$P := \{p \in [\pi_p]\}$
(iii)	$S^\circ := \bigcup_{f \in F} f^\circ - \bigcup_{h \in H} h^\circ$
(iv)	$S^- := \mathbb{R}^2 - S^\circ$
(v)	$\partial S := \overline{S^\circ} \bigcap \overline{S^-}$

Let $S = (F, H, L, P)$ be a structural region, we use the notation $C(S) = F \cup H \cup L \cup P$ to denote the set of all component objects in S.

An example structural region depicting each of the possible components is shown in Figure 3. Figure 3a depicts all structural components, and Figure 3b depicts the interpretation of the region (its interior, boundary and exterior). Note that since the components of a structural region are regular open point sets, and the interpretation of the interior, exterior, and boundary are based on set operations of the regular open point sets, the interpretation will always form a valid region [4, 10, 7]; this follows from the property that regular open point sets are closed under intersection, union, and difference.

Time is a set of *instants* such that each instant t is a real number. This representation allows access to the concept of a time instant, provides a total ordering on instants, and provides a continuous definition of time:

Definition 8 $T = \{t \in \mathbb{R}\}$

Structural regions are temporally static. To create a spatiotemporal type that allows the representation of a moving region, we must associate structural regions with a temporal component. *Interval regions* are defined as a pair of structural regions, each associated with a timestamp, such that the region associated with the earlier timestamp is the *source* and the region associated with the later timestamp is the *destination*. An interval region describes the motion of a region as it travels from the source to the destination over the time interval.

The movement of an interval region is defined by specifying the movement of each individual component of the source region as it travels over the interval to one or more components of the destination region. This movement is specified using two parts:

1. A *component mapping*: the component mapping maps a component from the source region to a component in the destination region. No restrictions are placed on the mapping, and any component in the source region may map to any component in the destination region.

2. A *movement function*: The movement function defines how a component from the source region progresses over the interval to arrive at its destination location and shape defined by the component in the destination region identified by the component mapping.

The lack of restrictions on the component mapping allows complex motion patterns to be represented easily. For example, a face in a source region may map to a hole in a destination region. The interpretation of this scenario, detailed later, will be that a face travels across a time interval, then disappears the instant it reaches the time of the destination region. Such movement may be practical in some applications (for example, a region representing a temperature above a certain threshold that disappears as soon as the temperature drops below the threshold).

As stated above, we assume the motion function takes a source and destination component as arguments, and creates a motion plan describing the motion of the source component across the interval. The motion function will return a 3-dimensional volume, with time as the third dimension, that represents this movement. Such 3-dimensional representations are common in moving object models. One important note about the movement function is that we assume that a single, general motion function is used for all structural regions in a particular system. In Section 5, we will define operations that require this assumption. This assumption represents a tradeoff between the ability to specify more specific movement in individual moving regions vs the ability to create general algorithms for operations that can take advantage of 2-dimensional algorithms on a 3-dimensional object. Note that a user can always introduce more explicit control over motion by creating more interval regions that each cover shorter time intervals.

Formally, the motion function will map instants in time to points, lines, and regions. Although the model does not explicitly require the mapping to result in a continuously moving object, many applications will have such a requirement.

Definition 9 *Let* $\alpha = [\textit{simple region} \cup \textit{simple line} \cup \textit{simple point}]$
A motion function is a mapping from time to components:
$m : T \rightarrow \alpha$
In practice, the motion function will be constrained to the temporal bounds of the interval region to which it applies:

$$f(s \in \alpha, t_1, d \in \alpha, t_2) := \left\{ \begin{array}{l} s \textit{ for } t = t_1, \\ d \textit{ for } t = t_2, \\ m(t) \textit{ for } t_1 < t < t2 \end{array} \right\}$$

The definition of the motion function is deliberately general to allow the adoption of motion functions for specific domains.

Equipped with the concepts of structural regions, component mappings, and motion functions, we are now able to defined interval regions. An *interval region* is a structural region associated with time. An interval region describes the motion of a structural region over a specified time interval. An interval region is defined by a structural region at the beginning of the time interval, denoted the *source region*, and a structural region at the end of the time interval, denoted the *destination region*. The interval region contains a start and an end time, respectively associated with the source and destination regions. The motion of the region across the interval is defined by the component mapping M. The component mapping in conjunction with a motion function define the movement of the region across the interval. The requirement that M be total and surjective insures that all components of the source and destination region exist across the time interval, and not simply at the boundary. The existence of components only at the boundary instants enforces all components within an interval to exist for the same amount of time. Furthermore, the rule that the start and end instants be different enforce temporal semi-continuity over a set of intervals; in other words, objects are not allowed to exist for only an instant, but must for some time interval, however small. This final requirement eases the definition of operations in later sections.

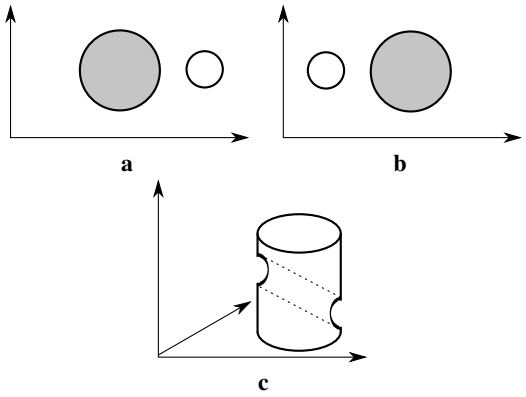

Figure 4: An interval region defined by a structural region (a) at time t_1 and a structural region (b) at time t_2. Each of the structural regions contains a shaded face and a hole. The hole will move across the face over the time interval. The interpretation of the interval region, determined by computing the interior, exterior, and boundary of the structural region at every time instant across the interval, is shown as (c).

Definition 10 An *interval region* is a pair of structural regions *(S, D)* $\in [\textit{structural region}]$ *and a mapping M, where*

(i)	$S := ((F_S, H_S, L_S, P_S), T_S)$
(ii)	$D := ((F_D, H_D, L_D, P_D), T_D)$
(iii)	$M := C(S) \rightarrow C(D)$
(iv)	$T_S, T_D \in T \textit{ and } T_S < T_D$
(v)	*M is total and surjective*

The definition of an interval region creates the ability to represent complex motion patterns in a simple manner. For example, a simple conception of a region in which a hole moves across the face of the region is complex to represent in other moving region models, especially at the implementation level. Because structural regions have a separate structural representation and interpretation, such a scene is easily represented in our model at the abstract level, and, we show later, in the implementation level. An example of such a scene and its interpretation is shown in Figure 4. Recall that an interval region will define a structural region at every time instant for which it is defined; the interpretation of the interval region is visualized as the interpretation of the structural regions at all time instants for which the interval region is defined.

Finally, a moving region is a set of interval regions:

Definition 11 *A* moving region *is a set of interval regions that may overlap at boundaries in time, but do not overlap in temporal interiors. Source time and destination time are inclusive.*

4. DISCRETE DATA MODEL

In this section, we provide discrete concepts required to implement the abstract model. In general, we use traditional discretization approaches for spatial types: the areal spatial types are represented as a collection of straight line segments representing the boundary of the spatial object, the linear types are represented as a

collection of straight line segments, and the point types are represented simply as their coordinates. As per convention, straight line segments defining a spatial object do not intersect in the interior of any line segment.

In order to define motion functions on discrete types, we require a type that represents the motion of a straight line segment across an interval. In the discrete realm, a motion function must produce the motion of a component of a structural region as it travels across a time interval defined by an interval region. We use the type of *moving segments* to represent such motion. We define a moving segment as a triple of points in 3-dimensional space (time is the third dimension) in which the the first two items form a straight line segment at a single time instant, and the third item forms a point. Therefore, a moving segment always describes the motion of a segment as it constricts across a time interval to a point, or vice versa. We do not specify a specific function to determine the movement of the line segment, since our model assumes a single movement function for all moving regions in a particular application, but the definition leaves the possibility for a choice of movement functions at the application level. Thus, we have:

Definition 12 *A* moving segment *is a triple:*
$(p = (x_p, y_p, z_p), q = (x_q, y_q, z_q), s = (x_s, y_s, z_s))$
where $p, q, s \in \mathbb{R}^3 \wedge z_p = z_q \wedge z_p \neq z_s$.
(p, q) *is a straight line segment, and s is a single point.*

A motion function is then a function that maps a source component and destination component to a set of moving segments that define the movement of the source component as it travels across the time interval to the destination component:

Definition 13 *Let* $\alpha = [$*simple region* $\cup$ *simple line* $\cup$ *simple point*$]$
Let [*moving segment*] *represent the type of all valid moving segments.*
A motion function mf *is a function that maps a source and destination component, respectively associated with time instants, to a set of moving segments:*
$mf : \alpha \times T \times \alpha \times T \rightarrow 2^{[\textit{moving segment}]}$

Again, it is expected that a motion function be a general function applied to all interval regions in a particular application setting; this is required for operational closure. For the remainder of the paper, we use the notation $mf()$ to indicate the application of a motion function. Basic motion functions exist [8, 12] that guarantee type closure across an interval.

5. ALGORITHMS

In this section, we provide algorithms to compute the geometric set intersection operation of intersection between two moving regions under the CMR model. We focus on utilizing well known 2D algorithms instead of more complex 3D algorithms. 2D algorithms are relatively simpler to comprehend and implement. Furthermore the algorithms are applied on snapshots of moving regions, which align well with data collection mechanisms. Only the intersection algorithm is included due to space limitations.

5.1 Align Moving Regions

All the algorithms defined in the following sections are binary, such that they take two moving regions as arguments. In general, these algorithms require that the argument moving regions have identical interval boundary times. Therefore, we need to "align" moving regions that do not have identical interval region boundary times before we proceed to apply set operations on them.

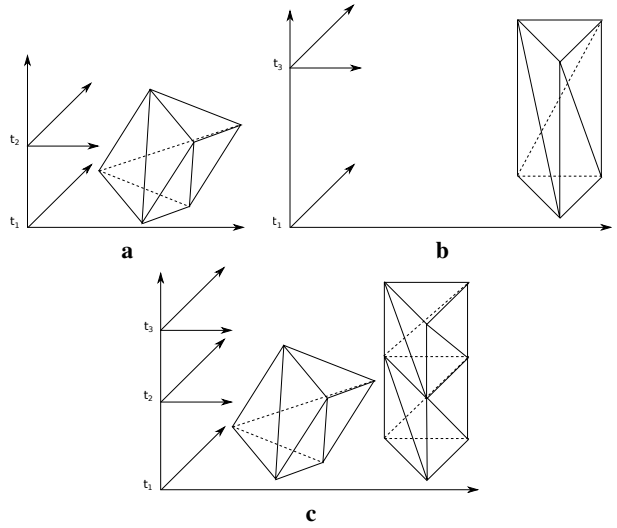

Figure 5: Two moving regions (a and b) that are not aligned. (c) The result of aligning (a) and (b).

The procedure to align two moving regions m_1 and m_2 begins by finding the set T of time instances, such that all interval region boundary times from m_1 and m_2 exist in T. We then insert interval region boundaries in m_1 for all times in T that do not correspond to an existing interval region boundary in m_1, and likewise for m_2. For example, in Figure 5, we align two moving regions, each containing a single interval region, by breaking one of the interval regions into two interval regions.

We define the *break* function to insert the temporal breaks into a moving region. The implementation is trivial given a set of moving segments:
$break : [\textit{moving region}] \times 2^T \rightarrow [\textit{moving region}]$

Algorithm 1 formalizes the alignment procedure. Two new moving regions, RM_1, RM_2, are obtained as a result of aligning two moving regions M_1 and M_2. Line 4 loops over all interval regions in input moving region M_1. The set of temporal boundaries for interval regions are recorded in line 5. Lines 6-7 repeat the procedure for M_2. The breaks are added to the input in regions in lines 8-9. The two moving regions are now aligned and ready for operations.

5.2 Intersection Operation

Recall that our goal is to use 2D operations instead of 3D operations whenever possible. Therefore, the intersection operation is defined in terms of intersecting two aligned interval regions. The general approach is to perform 2D intersection operations at the interval boundaries, then create a new mapping function for the resulting structures based on the mappings for the input interval regions. Again, this focuses on preserving known data at the snapshot boundaries, and allows flexibility in deciding movement between snapshots.

The algorithm to determine the intersection of two moving regions is summarized as follows:

1. At every point, where a moving segment from one region intersects with a moving segment from another region, we align both the moving regions.

2. For each interval region that has the same time boundaries as an interval region from the opposing moving region, we extract all of it's corresponding faces.

Input: Two moving regions, M_1 and M_2;
A set of boundary time values T corresponding to the temporal boundaries of all interval regions in M_1 and M_2
Output: Two aligned moving regions RM_1 and RM_2

1 $RM_1 = \{\}$;
2 $RM_2 = \{\}$;
3 $T' = \{\}$;
4 **foreach** $IR = (S, D, f, T_S, T_D) \in M_1$ **do**
5 $\quad T' = T' \cup \{T_S, T_D\}$;
6 **foreach** $IR = (S, D, f, T_S, T_D) \in M_2$ **do**
7 $\quad T' = T' \cup \{T_S, T_D\}$;
8 $RM_1 = break(M_1, T')$;
9 $RM_2 = break(M_2, T')$;

Algorithm 1: Aligning two moving regions.

3. Each extracted face of that interval region is intersected with similarly extracted faces of the opposing interval region, at the aligned time instant.

4. The same procedure is repeated at the other interval boundaries.

5. A new mapping is created to map the resulting structures across the intervals.

6. In case the region contains holes, we eliminate the holes that do not affect the faces obtained from the Step 3 and Step 4, and keep the holes that do affect the above intersection operation.

7. Subsequently, we update the mapping of the hole structures as well.

For example, Figure 6a shows two intersecting moving regions that are already aligned. This example is a simple case of regions without holes. The extracted faces for both the regions are intersecting at t_2. The intersection begins at the time t_{1-2} when the left triangle meets the right triangle, and continues until the left triangle ceases to exist at t_2; thus, the moving regions are also aligned at t_{1-2}. To compute the intersection, the temporally static regions are extracted at all aligned temporal boundaries. At each respective aligned temporal boundary, the static regions are then intersected, resulting in objects shown in Figure 6b. At this point, a mapping must be computed that indicates, for each source object in an interval, what its destination object will be at the opposing interval boundary (we provide more details in the full algorithm description that follows). In this simple case, the mapping is trivial because only two objects exist at each time boundary. Figure 6c depicts the result of the intersection with the motion function applied.

The complete algorithm for computing the intersection of two aligned interval regions under the CMR model is shown in Algorithm 2. Because moving regions are simply a set of interval regions, the intersection of moving regions is computed by computing Algorithm 2 for aligned pairs of interval regions. We walk through the algorithm using the example of computing the intersection of the interval regions shown in Figure 7. Figure 7 depicts two interval regions, one of which contains a hole. At time t_1, the darker shaded interval region intersects both the opposing region and overlaps the hole in that region. At time t_2, the darker shaded region has moved to the right such that it no longer overlaps the hole in the opposing region. For the remainder of this discussion, let IR_1 be the lighter shaded region in Figure 7a and IR_2 be the darker shaded region in Figure 7a.

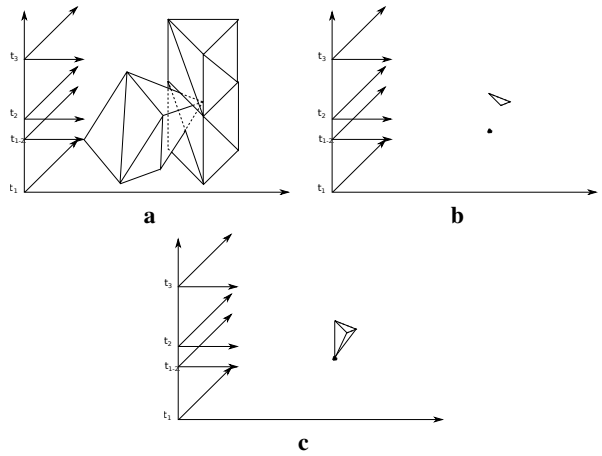

Figure 6: Intersection operation on two moving regions

The intersection algorithm begins by applying the movement function to each input interval region, resulting in a set of moving segments (line 4). A moving segment effectively defines a triangle in 3-dimensional space, thus, computing the intersection points in a set of moving segments is equivalent to computing the intersections in a set of triangles in 3-dimensional space. Furthermore, the semantically interesting intersections are the ones in which the endpoint of a moving segment crosses from the interior of the opposing region to the exterior, or vice versa; therefore, we are only interested in intersections involving a triangle edge. The temporal component of each such triangle intersection is recorded, and the input interval regions are re-aligned using this set of temporal values (line 6). In essence, we are computing the points in which topological changes occur between the segments of the input interval regions, the time in between the topological changes is interpolated using the motion function. In Figure 7, no such intersections occur, so the input is not required to be re-aligned. A naive algorithm to compute the intersections in a set of n triangles is trivial in $O(n^2)$ time.

Once the input interval regions are aligned, a set of interval regions exists. Each temporally corresponding pair of interval regions from each aligned input interval region are processed individually (lines 8-40). For an aligned pair of interval regions, the *extractFace* function is applied to compute the valid region faces that exist at the temporal boundaries of each interval region (lines 12-13). *extractFace* computes valid region faces by taking each face in a structural region and computing the difference of that face with the union of all holes in the structural region. The extract face function returns a pair for each valid face consisting of the valid face and the original face as it existed before the holes were removed. This computation provides two important pieces of data for the intersection algorithm: i) all faces that are not completely occluded by holes are computed, and ii) the original faces are maintained for use with the mapping function of the interval region. For example, Figure 8 depicts a face and hole, which are stored separately as part of a structural region, and the resulting region from applying the extractFace function.

$$extractface : Interval\ Region \times Time \rightarrow 2^{RegionXRegion}$$

Once the faces are extracted from the aligned interval regions, each pair of faces, respectively from each interval region, are tested for intersection (lines 16). In Figure 7a, the extracted faces at the earlier time boundary for IR_1 and IR_2 intersect, as do their extracted

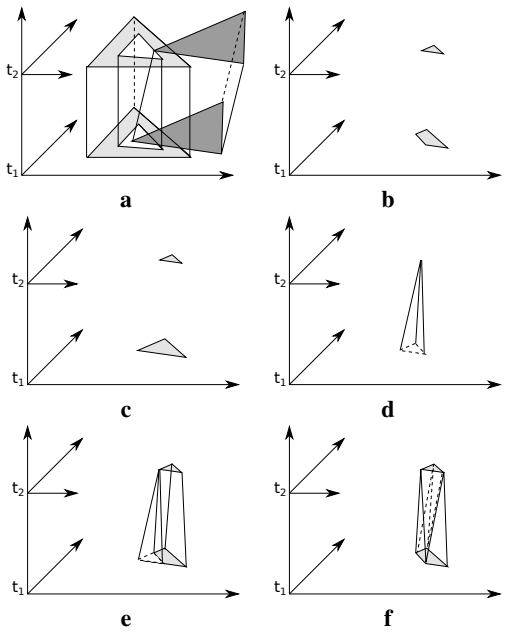

Figure 7: Two intersecting interval regions. The region faces are shaded at temporal boundaries to emphasize the presence of the hole.

faces at the later time boundary. If two extracted faces from respective regions intersect at the earlier time boundary of two aligned interval regions and the extracted faces corresponding to their mappings at the later time boundary also intersect (line 17), it follows that the region r corresponding to the intersection of those faces exists across the time interval (this test is in line 18). Clearly, in Figure 7a, the intersection of the extracted faces of IR_1 and IR_2 exists across the time interval. Thus, the existence of r must be recorded in the result interval region.

When recording that a face exists in the result interval region, it is important to recall that structural regions contain only simple regions as components, therefore, the intersection of two extracted faces cannot be directly recorded since it may contain holes or multiple faces. Instead, the original faces as they existed before extraction are stored (lines 19-20). Hole structures will be computed later in the algorithm to occlude the necessary portions of the faces. For example, Figure 7b shows the extracted faces of IR_1 and IR_2 that exist at the temporal boundaries of the result region, but the intersection of the original faces, shown in Figure 7c are actually stored in the structural regions of the result interval region. Note that the intersection of the original faces may contain multiple faces; in such a case, a decision must be made as to which face in the source region maps to which face in the destination region. Various methods of making this assignment exist, for example, assigning regions that overlap the most, but we leave the choice of method as a domain specific choice, and use the *assign* function to indicate that an assignment is made (line 21). Another important note is that the intersection of two faces may result in a point or a line. Such a case may indicate a face fading from existence over an interval, and is allowed. Finally, result interval region mapping is updated (line 23) and the resulting components are stored in the result interval region (lines 24-25).

In summary, the portion of the intersection algorithm that deals with faces proceeds by identifying pairs of faces from the earlier time boundaries of interval regions that both exist after extracting

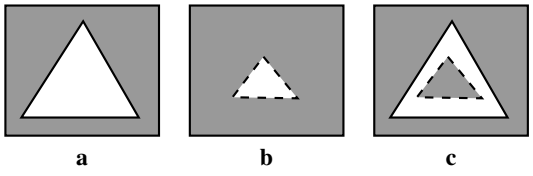

Figure 8: A face (a) and a hole (b) defining a structural region, and the result of computing the *extractFace* function (c).

them, that both intersect, and whose mapped structures at the opposing time boundary also intersect. Once such pairs are identified, the intersections of the original faces at each time boundary are computed, stored in the resulting interval region, and the resulting interval region's mapping is updated to reflect the motion of the faces across the interval region.

Once the faces of the resulting interval region are computed, the holes must be computed (lines 27-39). The process for computing holes in the result interval region is similar to that of computing faces, with two exceptions: i) each input region's holes are processed separately, and ii) the smallest portions of the input hole structures that are required to properly occlude portions of faces are kept in the result regions. Point (ii) differs from the handling of faces since faces in the result region are computed as the intersection of faces that are un-occluded by any holes (lines 19-20) .

A hole from an input interval region will only by included in the result interval region if it interacts with some face during the duration of its corresponding interval. Furthermore, only the portion of the hole that interacts with the face will be included in the final result. Thus, each hole at the earlier boundary of an input region, say IR_1, is tested to see if it interacts with one of the faces, or a point or line that maps to a face, from earlier boundary of the faces computed to be included in the result interval region (lines 30-31). A hole will only exist over an interval if it interacts with faces (or points and lines mapped to a face) at both time boundaries, so its mapped face at the opposing time boundary is also tested for intersection with the mapped face of any face with which it interacts (lines 32-33). For example, in Figure 7, the hole in IR_1 intersects a face in the result interval region, and the corresponding mapped hole at the later time boundary intersects (at a point) with the mapped face of the face with which it intersects at the earlier boundary.

Once a hole has been identified as belonging to the intersection of two interval regions, it must be stored in the result region. To determine that hole interacts with a face, the geometry defining the hole is intersected with the geometry defining a face; the result is possibly multiple geometries, each corresponding to a hole that occludes a portion of a face. Again, as with faces, multiple geometries may exist at both time boundaries for a hole, and a decision must be made as to which hole geometries map to which other hole geometries at the opposing boundary. Again, we use the *assign* function to make this decision, leaving the details as a domain specific decision (lines 34-35). The mapping of the result interval region is updated with the hole geometries (line 37), and the geometries are stored in their respective sets in the boundary structural regions (lines 38-39). The process is then repeated for holes in the other input interval region (line 41). Figure 7d shows the resulting holes from performing this process on IR_1 and IR_2. The resulting interval region contains the faces in Figure 7c and the holes in Figure 7d. The faces and holes shown together with their mappings are in Figure 7e. The interpretation of the resulting interval region is shown in Figure 7f.

The decision to keep portions of faces occluded by holes in the result interval region from an intersection is necessary since structural regions contain simple regions as faces, and thus, must have portions occluded by separate hole geometries rather than representing them as part of the face geometry. The decision to keep only the portions of holes that directly occlude a portion of a face in the result interval region of an intersection is required due to the interpretation of structural regions: the existence of faces add to the area covered by a structural region, but the existence of holes takes away area covered by the structural region. If a portion of a hole is kept that does not directly occlude a portion of a face, then a face added later by a user might unexpectedly have an occlusion from the portion of the hole that did not occlude any of the original faces.

One of the goals of the CMR model is to define algorithms that rely on 2-dimensional operations whenever possible. This goal is achieved in the moving region intersection algorithm; note that the only true 3-dimensional operations are the movement functions, which are available in the literature, a triangle/triangle intersection operation, a rather simple operation in 3-dimensions, and the function used to break an interval region into smaller intervals, again a rather simplistic operation. All other operations are 2-dimensional operations, including region/region intersection. Many solutions to the region/region intersection problem exist, many taking $O(n \lg n + k)$ time complexity, where k is the number of line segment intersections. Therefore, the major factors contributing to the overall time complexity of the moving region intersection algorithm are the 3D triangle/triangle intersection algorithm, and the nested *for* loops used to compute all pairs of faces from the input regions, and all pairs of holes from an input region and faces from the result interval region; a naive implementation of this functionality results in $O(n^2)$ time complexity, where n is the number of input line segments, for the nested loops and triangle/triangle intersection, respectively. Spatial partitioning techniques can reduce the time complexity of those portions of the algorithm $O(n \lg n)$ in many cases. Thus, the total time complexity for intersecting two moving regions with n input line segments is $O(n^2)$ for naive implementations, and $O(n \lg n) + k$ using spatial decomposition techniques.

6. EXAMPLE

In order to illustrate the power and ease of object representation in the CMR model, we end with an example using the illustrations in Figure 9. Assume a user is collecting rain cloud objects from radar images. At time t_1, a single rain cloud is detected (face (a) in Figure 9a). At time t_2 no rain clouds appear. Because the user is a domain expert, and has access to other data sources, the user discovers through weather stations that rain actually fell in a pattern over the time interval such that face (a) dissipated, eventually vanishing in a line structure (c) and a point structure (e). Therefore, the user adds the line and point structure to the structural region at time T_2, and updates the interval region mapping with $a \rightarrow c$ and $a \rightarrow e$. Effectively, a heat island-like effect in which a rain cloud splits is observed. Finally, the user notices that a hole in the rain cloud must have formed over the interval (again, by incorporating other data sources). The hole appears in the middle of the interval, begins small, and grows larger as it translates in space. The user then adds point (b) to the source region and hole (d) to the destination region, and updates the mapping with $b \rightarrow d$. This extremely complex motion pattern is easily described by a user who is not necessarily computationally proficient, but has expert domain knowledge. Furthermore, in the slice representation model, this scene would require multiple intervals, and the user cannot simply draw the beginning and end of the motion, but must draw many intermediate steps at each interval. In the case of Figure 9a, the full

Input: Two interval regions, IR_1 and IR_2, with identical time boundaries
Output: An interval region IR_r containing the intersection of IR_1 and IR_2

1 Let $IR_r = (S_r, D_r, M_r, T_{sr}, T_{dr})$;
2 Let $S_r = (F_{sr}, H_{sr}, L_{sr}, P_{sr})$;
3 Let $D_r = (F_{dr}, H_{dr}, L_{dr}, P_{dr})$;
4 Set $Tri \leftarrow mf(IR_1) \bigcup mf(IR_2)$;
5 Get $T \leftarrow$ *All intersection times of pairs of triangles in Tri*;
6 $M_1, M_2 \leftarrow$ align moving regions IR_1, IR_2 using Algorithm 1 with set of times T;
7 Get $T_{list} \leftarrow$ *A sorted list of set of times in T*;
8 **for** $(i = 0; i <\mid T \mid -1; i++)$ **do**
9 **if** *T(i), T(i+1) forms an interval* IR'_1 *in* M_1 *and* IR'_2 *in* M_2 **then**
10 $IR'_1 = (S_1, D_1, M_1, T_{s1}, T_{d1})$;
11 $IR'_2 = (S_2, D_2, M_2, T_{s2}, T_{d2})$;
12 X = $\{((Q_1, Of_1), ...(Q_n, Of_n))\} \leftarrow$ extractface(IR'_1, T_1);
13 Y = $\{((R_1, Oe_1), ...(R_n, Oe_n))\} \leftarrow$ extractface(IR'_2, T_1);
14 **foreach** $(Q, Of) \in X$ **do**
15 **foreach** $(R, Oe) \in Y$ **do**
16 A = intersect(Q, R);
17 B = intersect(extractface$(M_1(Of))$, extractface($(M_2(Oe))$));
18 **if** $A, B \neq \emptyset$ **then**
19 $A' = intersect(Of, Oe)$;
20 $B' = intersect(M_1(Of), M_2(Oe))$;
21 $C = \{(a_1, b_1), \ldots (a_n, b_n)\} \leftarrow$ $Assign(A', B')$;
22 **foreach** $(a, b) \in C$ **do**
23 $M_r(a) = b$;
24 Place a in its appropriate set in S_R (faces, points, or lines);
25 Place b in its appropriate set in D_R (faces, points, or lines);
26 Let $S_1 = (F_{s1}, H_{s1}, L_{s1}, P_{s1})$;
27 Let $D_1 = (F_{d1}, H_{d1}, L_{d1}, P_{d1})$;
28 Let $S_r = (F_{sr}, H_{sr}, L_{sr}, P_{sr})$;
29 Let $D_r = (F_{dr}, H_{dr}, L_{dr}, P_{dr})$;
30 **foreach** *hole* $h \in H_{s1}$ **do**
31 **foreach** *face* $f \in F_{sr} \cup \{g = M_r^{-1}(e \in F_{dr}) \wedge e \in P_{sr} \cup L_{sr}\}$ **do**
32 A = intersect(h,f);
33 B = intersect$(M_1(h), M_r(f))$;
34 **if** $A, B \neq \emptyset$ **then**
35 $C = \{(a_1, b_1), \ldots (a_n, b_n)\} \leftarrow$ $Assign(A, B)$;
36 **foreach** $(a, b) \in C$ **do**
37 $M_r(a) = b$;
38 Place a in its appropriate set in S_R (faces, points, or lines);
39 Place b in its appropriate set in D_R (faces, points, or lines);
40 Repeat previous loop using H_{s2} and M_2;

Algorithm 2: Algorithm for computing the intersection of two moving regions under the CMR model.

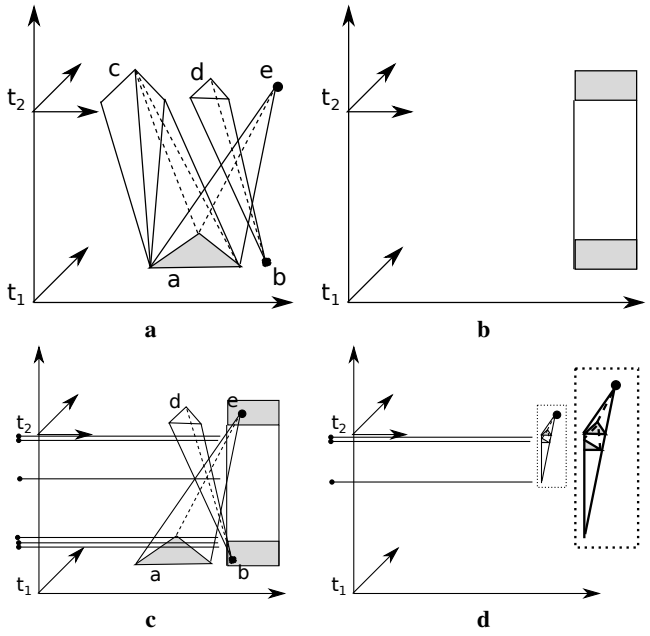

Figure 9: Two CMR model moving regions (a and b), their alignment instants (c), and their intersection (d). The shaded simple regions are faces, the non-shaded simple regions are holes.

interval region definition is:

$$S := ((F_S = \{a\}, H_S = \emptyset, L_S = \emptyset, P_S = \{b\}), t_1)$$
$$D := ((F_D = \emptyset, H_D = \{d\}, L_D = \{c\}, P_D = \{e\}), t_2)$$
$$M := \{a \to c, a \to e, b \to d\}$$

Finally, the user may wish to find the portions of the rain cloud that affects the county shown in Figure 9b. As per the intersection algorithm, the regions are aligned. Figure 9c shows the alignment instants, and only the structures involved in the intersection. Because the earlier alignment points only involve a hole and a face being intersected, nothing is retained in the result at those instants. Figure 9d shows the result of the intersection and the time instants of the resulting intervals. The view is also expanded for clarity.

7. CONCLUSION

In this paper, we defined the CMR model of moving regions at the abstract and discrete levels. The CMR model is designed from its inception to focus on the representation of moving regions at interval boundaries. This decision aligns with data collection mechanisms, such as imagery and video, which are based on timestamped images. The portions in between interval boundaries are interpolated using a general motion function. The result is that creating data for the CMR model is similar to creating temporally static regions; users can control the level of detail of motion by creating more or fewer interval boundaries. Furthermore, domain experts without computational expertise can create data objects without having to manually interpolate in between snapshots.

The decision to focus on interval boundaries in the CMR model has the added benefit of allowing operations to focus on interval boundaries as well. Thus, complex operations, such as moving region intersection, are reduced to the problem of finding when intervals should be introduced into a result region, extracting temporally static regions from input regions, and computing 2-d spatial operations on the extracted regions. Although only the intersection operation was included due to space limitations, union, difference, and topological predicates are computed similarly.

Future work includes implementing a full algebra on the CMR model, and extending the model to include moving lines and moving points.

8. REFERENCES

[1] José Antonio Cotelo Lema, Luca Forlizzi, Ralf Hartmut Güting, Enrico Nardelli, and Markus Schneider. Algorithms for moving objects databases. *The Computer Journal*, 46(6):680–712, 2003.

[2] Victor Teixeira de Almeida, Ralf Hartmut Güting, and Thomas Behr. Querying moving objects in secondo. *2013 IEEE 14th International Conference on Mobile Data Management*, 0:47, 2006.

[3] J. Dugundi. *Topology*. Allyn and Bacon, 1966.

[4] Martin Erwig and Markus Schneider. Partition and conquer. In StephenC. Hirtle and AndrewU. Frank, editors, *Spatial Information Theory A Theoretical Basis for GIS*, volume 1329 of *Lecture Notes in Computer Science*, pages 389–407. Springer Berlin Heidelberg, 1997.

[5] Luca Forlizzi, Ralf Hartmut Güting, Enrico Nardelli, and Markus Schneider. A data model and data structures for moving objects databases. *SIGMOD Rec.*, 29(2):319–330, May 2000.

[6] Ralf Hartmut Güting, Michael H. Böhlen, Martin Erwig, Christian S. Jensen, Nikos A. Lorentzos, Markus Schneider, and Michalis Vazirgiannis. A foundation for representing and querying moving objects. *ACM Trans. Database Syst.*, 25(1):1–42, March 2000.

[7] Mark McKenney and Markus Schneider. PLR Partitions: A Conceptual Model of Maps. In Jean-Luc Hainaut and et. al., editors, *Advances in Conceptual Modeling - Foundations and Applications*, volume 4802 of *Lecture Notes in Computer Science*, pages 368–377. Springer Berlin Heidelberg, 2007.

[8] Mark McKenney and James Webb. Extracting moving regions from spatial data. In *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '10, pages 438–441, New York, NY, USA, 2010. ACM.

[9] Nikos Pelekis, Babis Theodoulidis, Ioannis Kopanakis, and Yannis Theodoridis. Literature review of spatio-temporal database models. *Knowl. Eng. Rev.*, 19(3):235–274, September 2004.

[10] Markus Schneider and Thomas Behr. Topological relationships between complex spatial objects. *ACM Trans. Database Syst.*, 31(1):39–81, March 2006.

[11] A. Prasad Sistla, Ouri Wolfson, Sam Chamberlain, and Son Dao. Modeling and querying moving objects. In *Proceedings of the Thirteenth International Conference on Data Engineering*, ICDE '97, pages 422–432, Washington, DC, USA, 1997. IEEE Computer Society.

[12] Erlend Tøssebro and Ralf Hartmut Güting. Creating representations for continuously moving regions from observations. In *Proceedings of the 7th International Symposium on Advances in Spatial and Temporal Databases*, SSTD '01, pages 321–344, London, UK, UK, 2001. Springer-Verlag.

[13] M. F. Worboys. A unified model for spatial and temporal information. *The Computer Journal*, 37(1):26–34, 1994.