ABSTRACT

We provide an algorithm that can construct a valid moving region from snapshots of a region in time. Our algorithm is robust in the sense that given valid input regions, the generated moving region will define a valid region at every instant over the given time period.

1. INTRODUCTION

Many disciplines involve the management and analysis of spatial data that has an associated temporal component. This temporal component may simply be a time-stamp of when data was collected, or may describe something more complex such as time frame when data is valid. In many cases, the spatial and temporal components of scientific data are fundamentally related in the sense that a spatial object is moving or changing shape throughout a time period. Such data are typically referred to as spatio-temporal data or moving object data. Examples of spatio-temporal data include weather phenomena such as storm systems or hurricanes, migratory patterns of land or ocean animals, regions of air pollution that drift through the atmosphere, and regions representing rates of infection as a disease spreads. Effectively collecting, analyzing, and managing spatio-temporal data is vital to fields such as meteorology, biology, oceanography, and geography to name a few.

Despite the importance of spatio-temporal data, collecting it remains a challenging problem. Many sensor technologies exist and are readily available with the ability to capture spatial data and associate it with some temporal component. For example, radar, satellite imaging, and even video systems can be leveraged, but these systems have the common property that they effectively take snapshots of spatial data at a fixed point in time, and then associate a temporal value with each snapshot. A region at a specific point in time representing a hurricane, for instance, can be created from a satellite image; however, a representation of a hurricane that models its movement with infinite temporal resolution can more accurately model the continuous motion of the hurricane as compared to a series of temporally static snapshots. A continuously moving representation also allows for more precise analysis of the hurricane’s path, trajectory, and interference with land masses. Therefore, given a series of snapshots of a spatial object over time, a need exists to be able to construct a continuously moving spatio-temporal object from the snapshots. Specifically in the case of spatial data consisting of regions, we denote this the region interpolation problem (RIP), since the movement of a region must be interpolated between snapshots. The RIP proves to be a difficult problem because the moving region that results from an interpolation must be constructed in such a way that at every time instant, the temporally static, spatial region described by the moving region must conform to the definition of spatial regions. If at some time instant within a moving region, the region at that instant violates the definition of a spatial region, then the entire moving region is invalid and type closure properties cannot be maintained across spatio-temporal operations. Type closure properties are vital in applications such as GIS and spatial databases where large amounts of data may be processed with the output of operations used as input to other operations.

The main contribution of this paper is an algorithm takes two regions representing snapshots of a moving region at two time instants, and computes a moving region across the time interval. Furthermore, the algorithm is correct in the sense that the result will describe a valid region at every time instant within the interval. The algorithm is robust in the sense that given any two valid input regions, it will always return a valid result. A second contribution is that the proposed algorithm will at most divide a time interval into three smaller time intervals, effectively providing an upper bound on the storage requirements of the algorithm’s output. Finally, we are provide an example where existing approaches to solving the RIP fail due to assumptions about region interpolations that turn out to be false, and we show that our algorithm effectively handles such problematic cases.

2. RELATED WORK

Spatio-temporal databases have been studied in the literature for some time [7, 5, 2, 3]. The most comprehensive solution is presented as a mathematical model defining moving object data types, operations, and predicates in [2], and a discrete model more suited to implementation in [3]. In [3], a discrete representation of moving regions is called the slice representation. This representation essentially represents a moving region as a series of slices where each slice is a collection of moving line segments that describe the mo-
tion of the line segments defining a region’s boundary over a time interval. This is similar in concept to our notion of interval regions (Section 3), however, slices are more restrictive. Our goal is to develop a general algorithm that can be easily applied to various models of moving regions, and so we choose a less strict model that can be easily adapted to a more strict definition of moving regions.

The closest work to our algorithm is presented in [6], which attempts to provide a solution to the region interpolation problem. That solution is similar to ours in that it takes two simple regions (Section 3) as input, and not two regions in general. The main drawback to the algorithm in [6] is that it assumes that given any two simple region snapshots at times \( t_1 \) and \( t_2 \) as input, a single interpolation between those snapshots exists such that the interpolation describes a valid simple region at every time instant between \( t_1 \) and \( t_2 \). Section 4 provides an explicit example of input regions in which this assumption does not hold. Our algorithm provides a valid solution to the RIP even when such cases exist.

3. DATA MODEL

In this paper, we are concerned with creating valid moving regions that continuously represent the movement of a region over time given two simple regions as input. Simple regions are defined as regions that contain only a single face and no holes. For example, the country of Italy is not a simple region because it contains multiple faces (the mainland and the islands), and hole that does not belong to Italy where Vatican City lies. We refer to simple regions simply as “regions” for the remainder of the paper.

An implementation perspective, a region \( R \) is represented as a list of straight line segments in cyclic order. A segment consists of two end points, each consisting of an \( x \) and \( y \) coordinate: \( \text{point} = (x, y) \). A total ordering exists over points defined as: \( (x_1, y_1) < (x_2, y_2) \) \( \iff \) \( x_1 < x_2 \lor (x_1 = x_2 \land y_1 < y_2) \). A segment is then defined as a pair of points: \( \text{segment} = (p, q) \mid p \in \text{point} \land p < q \). For a line segment \( s \), the notation \( s_p \) and \( s_q \) respectively indicates point \( p \) and \( q \) in \( s \). A total ordering exists over segments as follows: given two segments \( s \) and \( u \), and the respective angles measured counter-clockwise between each segment and a vertical line emanating up from its least point, \( \theta_p \) and \( \theta_u \), then: \( s < u \iff (s_p < u_p) \lor (s_p = u_p \land \theta_p < \theta_u) \). Cyclic order is the order the segments will be visited in if a person walks along the boundary of the region beginning from the least most end point in the least most segment towards the greater end point in the least most segment and continues around the cycle (i.e., counter-clockwise around the cycle from the least most segment). When traversing a segment in cyclic order, one endpoint of a segment will be encountered before the other end point. We refer to the first endpoint encountered on a segment when traversing the region boundary in cyclic order as the primary point, and the other as the secondary point. We assume that the list of segments defining a region \( R \) is in cyclic order.

Intuitively, a moving region is defined as a mapping from time instants to regions [2] where \( \tau \) is the set of all time instants and \( A \) is the set of all valid regions (including the empty region): \( \tau \rightarrow A \). Therefore, at any given time instant, a valid region exists (even if that is the empty region). For implementation, we define a moving region to be represented by a series of temporally disjoint interval regions [3]. An interval region from time instant \( t_0 \) to \( t_1 \) is defined as a list of moving segments describing the motion of a segment beginning at time \( t_0 \) and traveling across the interval until time \( t_1 \). We assume that a moving segment always consists of a segment and a point; therefore, a segment will either contract to a point over the time interval, or a point will expand into a segment over the time interval. Thus, a moving segment consists of a segment, a point, and a boolean flag indicating if the moving segment begins the time interval as a segment or as a point (a true value indicating the moving segment begins the interval as a segment): \( \text{moving segment} = (s, p, b) \mid s \in \text{segment} \land p \in \text{point} \land b \in \{0, 1\} \). We assume that each endpoint of the segment in a moving segment will travel at a constant rate across the time interval; thus, the constant velocity trajectories of the endpoints of the moving segment can be viewed as lines connecting the endpoints of the line segment at one end of the interval with the point contained in the moving segment at the other end of the interval. We use the term interval boundary regions to describe the two regions defined by an interval region at either end of the interval. The interval boundary region at the earlier time is denoted the source region and the region at the later time is denoted the destination region. When plotting a moving segment in three dimensional space, with time as the third dimension, the end points of the segment and the point that define the moving segment form the vertices of a triangle. We denote such triangle delta triangles and use them as the plotting mechanism for interval regions.

4. ALGORITHM

Given two interval boundary regions and times corresponding to a time interval, the algorithm presented in this paper creates a bounded set of interval regions describing a moving region’s motion across the time interval. In some situations, only a single interval region is required to represent a region’s motion; however, certain configurations exist such that a single interval region will be invalid (i.e., there exists a time instant during the interval when the geometry defined by the interval region violates the definition of a simple region). Our algorithm detects such invalid configurations, and if none are found, the algorithm is complete. If an invalid configuration is found, then the interval region is split into at most three interval regions. We develop our algorithm in stages in which each stage addresses a more complex problem that must be solved.

Twisting: Given two convex simple regions at times \( t_1 \) and \( t_2 \), a solution to the RIP will be a set of moving segments that describe the motion of boundary line segments from the source region to the destination region. The set of moving segments essentially defines a mapping from segments in the source region to points in the destination region, and points in the source region to segments in the destination region describing their motion over the time interval. There are obviously many choices to construct this mapping, but a mapping must address two key issues. First, the resulting region must have a closed boundary throughout the interval; therefore, no gaps between the endpoints of adjacent moving segments must appear in a region. Second, moving segments cannot overlap, except along segment endpoints (i.e., delta triangle boundaries). If we consider the simple case of computing the solution to the RIP between two convex interval boundary regions, then delta triangles will only intersect if they form a twisting motion across an interval. For example, if the source and destination regions are identical squares,
delta triangles could be defined to travel from an edge to a corner of a square on the opposite side. If all edges are defined in this way, then delta triangles will intersect, resulting in an invalid moving region. We solve both problems (eliminating gaps and avoiding intersecting delta triangles) for convex regions by creating moving segments through mapping segments to endpoints around source and destination regions in cyclic order. Twisting is avoided through the mapping of points in an interval boundary region to segments in the opposing region that are roughly the same distance around the boundary of the regions in cyclic order. We make the observation that as one traverses a convex region boundary in cyclic order, the angle as measured counter-clockwise from a vertical line extending down from the primary point of a segment and the segment itself either increases or stays the same as each successive segment is encountered. Therefore, this angle can be used as a measure of how far around a convex region a segment lies. For a segment \( s \), we denote this angle the progress angle for \( s \) and identify it by the notation \( \phi_s \). In essence, the next segment used as the segment portion of a moving segment will be the next segment in cyclic order in either the source or destination region that has not yet been used to create a moving segment and that has the smallest progress angle. The point from the opposing region used to complete the moving segment will be the secondary end point of the segment last used as the segment portion of a moving segment. The algorithm begins with the least segment in the source and destination regions, then repeating this process of creating moving segments until all segments are used to create moving segments (Figure 1a).

**Non-Convex Regions:** To extend our solution to non-convex regions, we must add a mechanism to deal with concavities in regions. The use of a progress angle to measure relative distance around two regions does not directly extend to concavities. Furthermore, concavities can be arbitrarily complex. Because our goal is create a geometrically valid moving region in an efficient manner, we take the approach that all segments that form a concavity in a source region will be used to form moving segments with the same point in the destination region, and vice versa. Effectively, this implies that a concavity in a source region will always collapse into a single point in the destination region, and a concavity in the destination region will always expand from a single point in the source region (Figure 1b). Because topological relationships are invariant under the operations of translation, rotation, and scaling, it follows that moving segments forming a concavity will never intersect using this approach since they are simply being translated and scaled to a single point in space. Therefore, an individual concavity in a source or destination region will not contain intersecting delta triangles, and will not violate the definition of a region at any time instant during an interval. Because all segments forming a concavity are dealt with as a group, the convex hull of a non-convex region is used to compute progress angles and prevent twisting. Because cyclic ordering is employed, we use Melkman’s convex hull algorithm to achieve a convex hull in linear time [4]. Algorithm 1 provides the solution to RIP as discussed thus far. A detailed discussion of the algorithm must be omitted due to space constraints.

**Intersecting Concavities:** Algorithm 1 creates an interval region such that invalid configurations do not arise from intersections involving segments forming a single concavity, nor do they form from twisting configurations due to the use of progress angles on the convex hull of a region; however, the delta triangles forming two separate concavities may intersect. Recall that intersecting delta triangles translate to intersecting moving regions, which implies a self-intersecting region boundary and an invalid region configuration. Therefore, we must be able to detect and handle intersecting concavities. Because delta triangles can be treated as triangles in 3-dimensional space, a triangle-based surface interference algorithm can be used to detect intersecting concavities. We chose an OBBTree based interference algorithm [1] due to its \( O(n \log n) \) construction time its ability to create bounding boxes oriented to the geometries of objects they bound; thus greatly reducing bounding box overlap among objects that do not intersect. The only modification required to the algorithm is that it should not report delta triangles as overlapping if they share an endpoint. This modification comes from the observation that many situations exist in which delta triangles can share vertices, but two delta triangles that overlap within their interiors will never share a vertex due to the generation of moving segments in cyclic order, the use of progress angles to prevent twisting, and the fact that all segments forming a single concavity in a regions are mapped to the same point in the opposing region. We were able to add roughly ten lines of code to a freely available implementation of an OBBTree interference detection algorithm to achieve this behavior.

Once intersecting concavities are detected, they must be dealt with. Our approach is to discover offending concavities, and remove them. Two or more concavities from a source region may intersect if they form a configuration such that a point exists in one concavity that is completely surrounded by a second concavity. This configuration typically occurs when concavities form spirals that often resemble the shape of a snail shell, thus, we denote it the snail

**Figure 1:** The interval region computed with convex region input (a) and with one non-convex region as input (b). Two examples of the snail shell configuration (c and d), and the series of interval regions computed between them (e).
shell configuration (Figure 1c). Because moving segments cannot change direction over an interval, the spirals in a snail shell cannot be unwound in single interval while preserving a valid geometry. Furthermore, unwinding such a configuration would require that an interval be split into an arbitrary number of smaller intervals since spirals can have an arbitrary number of turns. Therefore, our approach is to simply remove offending concavities in a single interval. An offending concavity in a source region is triangulated, and the triangles are simply collapsed towards an interior point over an interval. We denote this process evaporation, and its reverse (for offending concavities in a destination region), condensation. Because concavities are evaporated in place, all offending concavities can be evaporated (or condensed) in a single interval; therefore, a single interval between a source and destination regions must be divided into at most three intervals, one for evaporation, one for condensation, and one to transition the post-evaporation region to the pre-condensation region (Figure 1e). Note that if two cavities intersect from opposing regions, one must be chosen for removal in an evaporation or condensation stage. Note that the creation of a moving region from a source region that exhibits a snail shell configuration to a destination region cannot be achieved in a single interval using moving segments that do not change direction within an interval. This example violates the assumption required by the algorithm in [6] that only a single interval is required to create a moving region between arbitrary source and destination regions.

5. CONCLUSION

In this paper, we have provided a solution for the region interpolation problem for moving regions. Our solution is robust in that there are no special cases in which it fails, and it is efficient. In the cases where the interval boundary regions are convex, our algorithm has time complexity $O(n)$. If non-convex regions are used, the algorithm runs in $O(n \log n + (mn)^2)$ where $n$ is the number of line segments and $m$ is the number of concavities in the interval boundary regions. The $n \log n$ component reflects the building time of an OBBTree and the $(mn)^2$ reflects the worst case behavior of finding intersecting concavities in which each concavity must be tested for intersection against all other concavities. In practice, the worst case behavior is extremely rare, and only occurs in cases where the bounding boxes of all delta triangles intersect, but no actual delta triangles intersect; a situation that OBBTrees specifically minimize. Furthermore, two concavities intersect if only a single pair of delta triangles intersect, so the algorithm can be cut short when an intersecting pair is found. $m$ is also typically much smaller than $n$. We found through experiments that these factors result in extremely fast execution times despite a poor worst case complexity.

6. REFERENCES