Geometric and Thematic Integration of Spatial Data into Maps

Mark McKenney
Department of Computer Science,
Texas State University
mckenney@txstate.edu

Abstract

The map construction problem (MCP) is defined as a spatial data integration problem relating to the integration of region data into map data. Although a purely geometric integration of regions into a map is known and is efficient, algorithms preserving thematic data of regions are much more difficult. A naive approach to the MCP runs in $O((nm \lg nm)^2 + k)$ *time for m regions composed with* n *line segments on average with* k *line segment intersections. A new* $O((n+k)(\lg n + m + \lg m^2))$ *algorithm is presented to solve the MCP. The algorithm has been implemented and experiments show that it is significantly faster than the naive approach.*

Keywords: Geographic information systems, algorithms, data management, spatial data structures.

1. Introduction

The traditional basic units of spatial representation are the *point, line, and region*. Points represent collections of points in space, lines are used to represent one dimensional features and networks, such as roads and rivers, and regions represent areas, such as states or countries. These spatial primitives form the foundation of spatial storage techniques; however, spatial visualization typically occurs in the form of *maps*. Maps are a familiar concept that incorporate spatial, thematic, and even temporal data into an intuitive representation that inherently manages large amounts of data in a visual form: for example, adjacency of regions, connectivity of road networks, and the relationship between thematic data and spatial and geographical phenomena are easily identified in maps. Despite the utility of maps, maps are represented, stored, and managed as collections of their component points, lines, and regions in spatial systems. For example, a map of the United States may show the fifty states as a single map, but in current storage approaches, fifty separate regions representing the states will be stored, not a single map. Therefore, the information inherent in a map is not represented and must be explicitly computed from a collection of spatial primitives.

The use of maps as primarily a visualization tool in spatial systems leads to two problems: (i) maps are difficult to reuse in forming new maps or in computing operations over maps because they are not a data item in themselves, but a visualization of many separate data items, and (ii) information that is naturally encoded into maps, such as adjacency of items, must be computed on demand because no relationships are maintained among the separate constituents of the map since they are independent data items. We propose a new concept of maps in spatial systems in which maps are individual data items, not conglomerates of separate data items. This new concept of maps provides three benefits: (i) visual information and spatial relationships inherent to map structure are automatically maintained, (ii) map reuse is facilitated because operations to manipulate, extract information from, and combine maps as individual data items are well documented [9, 10, 3], and (iii) algorithms for maps can be expressed over single map data items instead of collections of spatial primitives, which allows for much more efficient algorithm implementations enabling map management on a large scale. However, the creation of map data from region data is problematic.

The *map construction problem* (MCP) is the problem of creating a single map data item from a collection of individual regions. The MCP is essentially a spatial integration problem in which separate spatial components and their associated thematic information must be preserved in the context of a more complex data item representing an entire map. In this paper, we consider maps that contain only region structures, leaving point and line structures to future work. Given a collection of m regions, a mechanism is required to construct a single data item that represents the map defined by the collection of regions. From a spatial and geometric perspective, mechanisms exist to combine separate region items into a single partition of space [1]; however, maintaining the associated thematic data with portions of regions that overlap in a map is not addressed. Currently, a

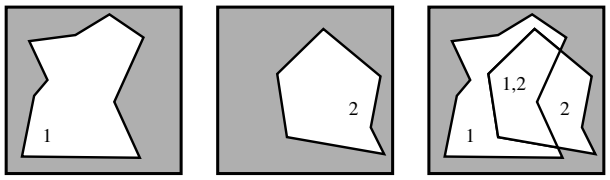

Figure 1. Two labeled regions, and the map formed when they are overlaid.

map must be computed incrementally by repeatedly adding a single region to a growing map. For m input regions, this results in $O((mn \lg mn)^2 + k)$ complexity, where n indicates the average size of a region in terms of its representation and k is the number of intersections among boundary line segments. We propose a new integration mechanism that can achieve a map from a collection of m regions in $O((n+k)(\lg n + m + \lg m^2))$ time. An implementation of the two techniques confirms our mechanism is significantly faster.

2 Preliminaries

The traditional spatial data type of complex regions [8] describes a region as a collection of *faces* such that faces must be disjoint or meet at a finite set of points. A face can contain zero or more *holes*, describing area that is not part of a face, but surrounded by it. For example, Italy contains multiple faces, its mainland and islands, and a hole that does not belong to Italy where Vatican City lies. From an implementation perspective, regions are typically represented by a set of straight line segments indicating the boundary of the region (Figure 1).

In this paper, we refer to maps as defined in the model of spatial partitions [7, 3]. We choose this model because it is mature, and a full algebra is defined around it, including operations and predicates. In short, a spatial partition defines a collection of regions such that regions either meet along boundaries, or are pairwise disjoint. Each region is assigned a unique *label*, which models thematic information. Regions are not allowed to overlap. If two regions r and s do overlap, their intersection is computed and three regions are actually inserted into the map, the intersection of r and s, and the differences $r - s$ and $s - r$. The label of the intersecting portions of the regions will contain the labels of both regions; thus, three *region primitives*, each with unique labels, represent the original two regions. Figure 1 depicts and example.

The combination of the geometric portions of regions in 2-dimensional space into a map is achieved through line segment intersection algorithms. In this paper, we employ a version of the *plane sweep algorithm* for computing overlays of regions and maps [1, 5]. Given two input regions, their overlay consists of both regions overlaid on each other such that boundary line segments intersect only at end points. The plane sweep algorithm proceeds by sweeping an imaginary line over a pair of regions or maps, R and W. Each time the line encounters a new line segment on the boundary of one of the input geometries, that segment is added to a list of segments actively being considered, known as the *active list*. Each time the line moves past a segment, that segment is removed from the active list. The active list stores the line segments in the order in which they intersect the sweep line; thus, if the sweep line is traveling in the x direction of a euclidean plane, the segments in the active list are sorted based on the y value of their intersection point with the imaginary sweep line. Because segments in the active list are sorted, the segments in the active list adjacent to a segment s from region R that is newly inserted into the active list contains the information necessary to determine whether s lies on the interior, boundary, or exterior of W. This knowledge is deduced because each line segment carries two identifiers, one indicating the region that lies above the segment, and the other indicating the region that lies below the segment. If no region lies above or below a segment, then a default label indicating the exterior of the region is assigned. The plane sweep algorithm *processes* one segment in each iteration of the algorithm. Processing a segment involves discovering if it intersects any other line segments, adding it to the active list, and determining which region interiors lie above and below the segment by looking at its immediate neighbors in the active list.

In implementation, the sweep line does not move continuously in a sweep line algorithm, but instead progresses based on segment endpoints. Therefore, each segment is actually represented twice, once for its beginning end point, and once for its ending end point, and the segments are sorted. Because segments are represented twice, the notion of a *halfsegment* is used to represent them. We define the type $halfsegment = \{(s, d, l, r) | s \in segment, d \in bool, l, r \in \mathbb{Z}\}$ where a segment is a straight line segment between two endpoints and l and r are *labels* corresponding to the portion of the embedding space that lies above or to the left of the line that the halfsegment lies on, and the portion that lies below and to the right, respectively. Thus, a halfsegment is said to have two *sides*, a left and right side corresponding to each label. As a matter of notation, we refer to l as the *above label* of h, and r as the *below label* of h. For a halfsegment $h = (s, d, l, r)$, if d is true (false), the smaller (greater) endpoint of s is the *dominating point* of h, and h is called a *left* (*right*) *halfsegment*. Hence, each segment s is mapped to two halfsegments $(s, true, l, r)$ and $(s, false, l, r)$. Let dp be a function which yields the dominating point of a halfsegment and len be a function that returns the length of a halfsegment. For two distinct half-

segments $h_1 = (s_1, d_1, l_1, r_1)$ and $h_2 = (s_2, d_2, l_2, r_2)$ with a common endpoint p, let α be the enclosed angle such that $0° < \alpha \leq 180°$. Let a predicate $rot(h_1, h_2)$ be true if, and only if, h_1 can be rotated around p through α to overlap h_2 in counterclockwise direction. We define a total order on halfsegments as:

$$
\begin{array}{l}
h_1 < h_2 \Leftrightarrow \\
dp(h_1) < dp(h_2) \vee \\
(dp(h_1) = dp(h_2) \wedge ((\neg d_1 \wedge d_2) \vee \\
(d_1 = d_2 \wedge rot(h_1, h_2)) \vee \\
(d_1 = d_2 \wedge collinear(s_1, s_2) \wedge len(s_1) < len(s_2))))
\end{array}
$$

Figure 2 depicts an example sequence of the sweep line algorithm. In Figure 2, part of the sweep line algorithm is shown in which two triangles are combined into a map. Line segments are shown with their labels on each side. The sweep line is dotted, and the segments currently in the active list are shown dashed. The sweep line visits halfsegments in halfsegment order. The labels of all halfsegments behind the sweep line are finalized, anything in front of the sweep line must still be processed. Note the final step depicted: when the halfsegment is added to the sweep line, it is clear (based on the labels of the segment being added and the labels of the segment below it in the sweep line) that the interiors of both triangles intersect above the segment. This intersection of triangle interiors is indicated in the labels of the segment. Therefore, the sweep line progression provides the ability to compute the necessary geometric information to integrate regions into maps. Furthermore, the opportunity to integrate thematic information exists, but the complexity of handling thematic information can be dramatic since an arbitrary number of regions may intersect.

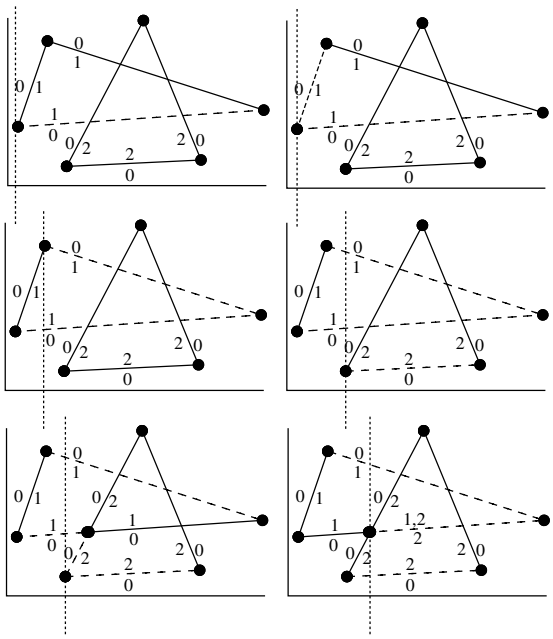

Figure 2. A partial example sequence of a sweep line algorithm integrating two triangles into a map. Segments are shown with labels indicating where triangle interiors lie. Note that only labels to the left of the sweep line are finalized. The label 0 indicates the exterior of both triangles.

3 Integrating Regions Into a Map

In this section, we describe our proposed algorithm that integrates regions and their associated thematic values into a map form that maintains all spatial and thematic data from the input regions. The input to our algorithm is a set of regions. We assign each region a label in the form of a unique integer identifier, known as the *region identifier*, that will be used to link each region with its thematic data. We then prepare the regions for input to the algorithm.

3.1 Preparing the Input

As mentioned previously, the plane sweep algorithm can detect line segment intersections, and determine whether the currently processed line segment lies in the interior, boundary, or exterior of other regions. The input to the plane sweep algorithm is a set of halfsegments sorted in halfsegment order. Our version of the algorithm requires that each halfsegment carry two labels, indicating the identifiers of all regions that respectively lie immediately above and below the halfsegment. Initially, the halfsegments forming an individual region will each have the region identifier for that region on one side, and the region exterior label on the other. Some data representations will only store segments, or halfsegments that do not carry labels; in this case, the halfsegments can be generated and labels assigned to the appropriate sides of the halfsegments in $O(n \lg n)$ time for each input region of n halfsegments on average [6]. Once the halfsegments for all input regions are prepared, they must be sorted into a single list. For m regions containing n halfsegments on average, sorting takes $O(mn \lg mn)$ time.

3.2 Spatial Processing of a Halfsegment

Once the algorithm input is computed, the plane sweep portion of the algorithm commences. We assume a sweep line that travels from left to right across the euclidean plane in the x direction. The plane sweep portion of the algorithm serves two main functions: (i) detect and remedy line segments that intersect at points other than end points, and

(ii) merge the labels of the intersecting portions of regions. These functions are managed simultaneously within the algorithm. Recall that the plane sweep algorithm proceeds by processing a single halfsegment at a time. At each step of the algorithm in which a left halfsegment h is being processed, h is inserted into the active list, and the immediate neighbors of h in the active list are computed. These neighbors are a and b, the neighbor halfsegment that lies above h and below h in the active list, respectively. The intersection points between h and a, and h and b are computed, the segment $s \in \{a, b\}$ that forms the least intersection point p with h in halfsegment order is chosen. h and s are removed from the active list, split according to p, and the resulting segments that are less than h in halfsegment order are inserted back into the active list. This step is the traditional computation of line segment intersections performed by the sweep line.

3.3 Thematic Processing of a Halfsegment

Once the spatial portion of the processing of halfsegment h is complete, the second function of the algorithm proceeds. The goal of the second part of the algorithm is to maintain thematic information associated with regions in the presence of multiple overlapping regions. Recall that if regions overlap in the spatial partition map model, then the overlapping portions of the regions carry the labels of all the overlapping regions. We proceed with this discussion using examples. Consider Figure 4a in which two regions overlap. The region's have identifiers which we assume are reflected in the halfsegment's labels. The segments in the figure are labeled. For figures where segments are labeled, we use the notation 1_l and 1_r to indicate the left and right halfsegments corresponding to segment 1, respectively. The plane sweep begins and processes halfsegments $1_l, 2_l, 2_r$, and 3_l, all of which belong to the same region and do not intersect any other halfsegments. When the plane sweep algorithm begins to process 5_l, the active list contains 1_l and 3_l; 1_l is the neighbor below 5_l and 3_l is the neighbor above 5_l in the active list. It follows from halfsegment ordering that the above label of the halfsegment below 5_l in the active list will indicate which region 5_l lies inside, or if it lies in the exterior of all other regions. Therefore, 5_l lies in the interior of region 1 because the interior of region 1 lies above 1_l, and this will be reflected in its above label. Because the interior of region 2 lies above 5_l, it follows that the interiors of both region 1 and region 2 lie above 5_l. From the spatial partition definition, we follow the convention that overlapping portions of regions receive the labels of all regions involved in the overlap; therefore a new label is assigned to the overlapping portion, and the labels of 5_l must be changed to reflect the new topology (label management details will be discussed below). When the plane sweep algorithm processes 4_l, it will only have one neighbor in the active list: 5_l. An intersection is detected between the two, and the segments are split. Because no segments exist in the active list below 4_l, it cannot lie in the interior of another region, and its labels remain unchanged. If the overlapping portion of the region 1 and 2 receives the identifier 3, then Figure 4b indicates the result of the algorithm. Because the label 3 indicates the overlapping portion of regions 1 and 2, some record keeping must exist to indicate that region identifier 3 is equivalent to the pair of region identifiers 1 and 2.

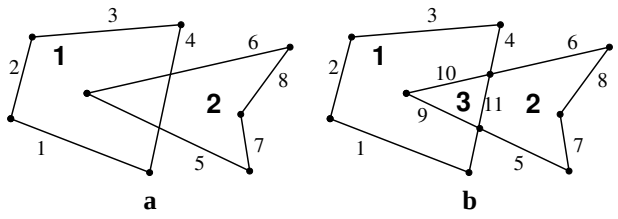

Figure 3. Two regions superimposed (a), and the result of their integration into a map (b). Segments are labeled.

Managing the labels of a pair of overlapping regions, as in the example above, is relatively straightforward; however, difficulty arises in cases where multiple regions overlap. For example, in Figure 4 three regions are shown, all of which overlap other regions. Again, computing the line segment intersections is straightforward, but managing region identifiers becomes much more challenging. For instance, when segment 8_l is processed, the halfsegment below it in the active list is 5_l. In order for us to correctly label the above label of 8_l as the overlap of regions 1, 2, and 3, the above label of 5_l must indicate that its above label is the overlap of regions 1 and 2, or we must proceed further down the active list than the halfsegment immediately below the one currently being processed. The time complexity bounds of the plane sweep algorithm rely on the fact that only the immediate neighbors of a segment in the active list are examined when processing the current segment, so we cannot look beyond the immediate neighbors without causing an $O(n^2)$ time complexity. Therefore, we must devise a mechanism such that when a halfsegment h is processed, all regions whose interiors intersect h can be identified by examining only the halfsegment below h in the active list. In essence, the labels of h must be merged with the labels of the halfsegment below h in the active list to reflect the overlapping regions; we denote this *label merging*.

A successful label merge of a halfsegment results in the halfsegment's labels indicating the identifiers of all regions that lie immediately above and below the halfsegment. Therefore, the label of a halfsegment must be a set of region identifiers indicating all regions whose interiors lie

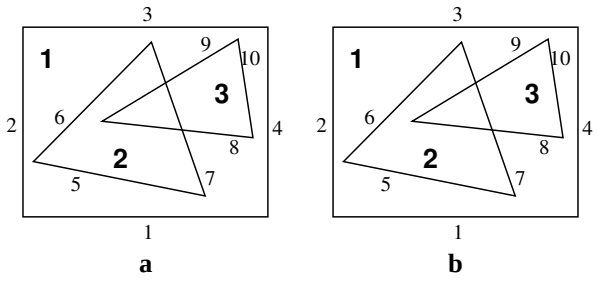

Figure 4. Three overlapping regions with labeled segments.

on either side of the halfsegment. For example, when halfsegment 8_l in Figure 4a is processed, it must indicate that the interiors of regions 1 and 2 lie immediately below the halfsegment, and the interiors of regions 1, 2, and 3 lie immediately above it. The original above and below labels of halfsegment 8_l are 3 and 0, respectively, where 0 indicates the exterior of the region. When 8_l is processed, we can determine the identifiers of all regions that must be involved in labels of 8_l by looking at the above label of the halfsegment below it in the active list, 5_l. Because 5_l is less than 8_l in halfsegment order, it will be processed before 8_l and its labels will be known when 8_l is processed. The above label of 5_l must indicate that the interiors of regions 1 and 2 lie above it. Therefore, the identifiers for regions 1 and 2 must be merged into 8_ls labels, indicating that 8_l lies within regions 1 and 2. When 9_l is processed, the halfsegment below it in the active list, 8_l, indicates that region identifiers 1, 2, and 3 must be merged with the labels of 9_l. Because the labels of 9_l indicate that it bounds region 3, the identifier for region 3 will not be added to the above label of 9_l (i.e., the interior of region 3 does not extend above 9_l). Therefore, the above and below labels of 9_l must indicate the region identifiers 1 and 2, and 1, 2, and 3, respectively. Formally, a label merge is represented as the following:

Definition 1 *let $h = (p, q, A, B)$ be a halfsegment and $h_b = (p_b, q_b, A_b, B_b)$ be the halfsegment below h in the active list when h is being processed by the plane sweep algorithm. Because h is not processed, it will have the identifier of the input region it bounds as one of its labels, and the other will be the identifier of the exterior. A* label merge *is then:*

$A = A \cup (A_b - B)$

$B = B \cup (A_b)$

These formulas follow directly from the definition of regions and the fact that a boundary always separates the interior of a region from the exterior of a region.

When a halfsegment is being processed, it will always have exactly one identifier in one of its labels, and the exterior identifier in the other label. The halfsegment below it in the active list may have many region identifiers in its above label. It follows from the definition of halfsegment merging that computing the below label of the current halfsegment h involves removing a single region identifier from the above label of the halfsegment below h in the active list. Furthermore, removing the below label of h from the above label of the halfsegment below h in the active list and inserting the above label of h also involves removing and adding single region identifiers. Therefore, the label merges can be expressed as three separate operations in which a single region identifier is added or subtracted to a set of region identifiers. When integrating m regions, the size of the set of region identifiers on the halfsegment below the halfsegment being processed is bounded by m; therefore, efficient insertion and removal operations are required. Thus, we assume these sets of region identifiers are implemented as a height bounded binary search tree which supports removal and insertion in $O(\lg m)$ time.

If many regions overlap in a map, then a label may contain many region identifiers. Storing a list of region identifiers for each segment forming the boundary of a map is inefficient in terms of space, especially when a map must be stored on disk; therefore, we maintain a mapping that defines a relationship between *label identifiers* and labels. A label identifier is a unique identifier that corresponds to a label. Because a label may contain many region identifiers, it is bounded by the number of regions being integrated. A label identifier can be implemented as a single integer, which is much more storage efficient and fits with the definition of halfsegments which specifies that labels are single integers, and not sets of integers. Again, a height bounded binary search tree is a good candidate for implementing this mapping and maintaining $O(\lg m^2)$ time complexity for adding, removing, and looking up elements. $O(\lg m^2)$ behavior is the worst case in which all possible combinations of overlapping regions exist, which is rare in practice. A reverse mapping from labels to label identifiers is also required and can be implemented efficiently using a trie, providing insertion and lookup in $O(m)$ time for m input regions [2, 4].

When a halfsegment h is being processed by the sweep line algorithm, the halfsegment below h is computed and its label is found. That label is copied twice, and a label merge is performed with the label for the region above h and below h respectively with each copy. The result of the label merges are labels indicating all regions whose interiors lie immediately above and below h. Because many halfsegments may contain identical labels, and these labels may contain up to m region identifiers for m input regions, representing each label with a label identifier can drastically reduce the memory requirements of the algorithm in situations when labels contain many region identifiers. Therefore, a trie associating labels with label identifiers is queried to see if the computed label is already assigned to another

halfsegment and already has an associated label identifier. If the label has not yet been encountered, then a new label identifier is created for the label, and the trie is updated. A height balanced binary tree is also updated to reflect the mapping from the label identifier to its corresponding label. Therefore, for each halfsegment visited by the plane sweep algorithm, a trie must be queried and possibly modified twice, and a height balanced binary tree must be queried and possibly modified twice, leading to a time complexity of $O(m + \lg m^2)$ for m regions for each halfsegment visited by the plane sweep. The optimal plane sweep has a time complexity of $O(n \lg n + k)$ for n segments with k intersections; however, the version of plane sweep that is used more typically in practice (and the version we use for implementation) has complexity $O(n \lg n + k \lg n) \Leftrightarrow O((n+k) \lg n)$ where $n + k$ segments are visited and the $\lg n$ term reflects operations on data structures necessary for each segment visited by the algorithm. Therefore, the overall time complexity of the algorithm is $O((n + k)(\lg n + m + \lg m^2))$. The space complexity of the algorithm depends on the number of unique labels created during the integration. In the worst case, every region will overlap every other region in such a way that m^2 labels will exist for m regions. Because each unique label is stored, and every segment from regions must be stored, the space complexity is $O(m^2 + n + k)$ for m regions containing n line segments with k intersections. In practice, the number of unique labels generated is typically much less then m^2; furthermore, m is typically much smaller than n, especially in geographic data sets, so a larger space or time complexity on the m term is acceptable.

We have implemented the region integration algorithm described in this paper, as well as the traditional $O((nm \lg nm)^2 + k)$ algorithm, in C++. Although the proposed algorithm is clearly faster in terms of algorithmic complexity, a practical comparison verifies the utility of the new algorithm. We ran the new and traditional algorithms over geographic data sets consisting of the counties of Florida, Alabama, and Texas. The input counties were taken from freely available US Census data sets, and the counties for each state were integrated into a single map of the state. The algorithms were run on a computer with a 2.8GHz processor and 2GB of RAM. Table 1 shows the running times of the algorithms, and the sizes of the resulting maps. Our new algorithm is clearly more efficient than the traditional method, and scales much better for larger input.

State	Halfsegments	New Algorithm (s)	Traditional Algorithm (s)
FL	129,718	12.291	80.295
AL	159,346	16.891	149.319
TX	517,938	60.954	1095.028

Table 1. Running times for the new algorithm and the traditional algorithm for constructing maps consisting of the counties of Florida, Arizona, and Texas. The number of halfsegments in each resulting map is shown. Each time is the average of three runs of the respective algorithm for the given data.

4. Conclusion

In this paper, we have introduced the map construction problem as a data integration problem requiring the handling of both geometric and thematic aspects of spatial data. We proposed a new algorithm that solves the map construction problem for m regions containing n line segments on average in $O((n + k)(\lg n + m + \lg m^2))$ time complexity and $O(m^2 + n + k)$ space complexity. This algorithm creates new opportunities for the use of maps as reusable data items, rather than simply visualizations. Future work includes the extension of the proposed algorithm to include spatial point and line objects.

References

[1] J. Bentley and T. Ottmann. Algorithms for Reporting and Counting Geometric Intersections. *IEEE Trans. on Computers*, C-28:643–647, 1979.

[2] R. de la Briandais. File Searching Using Variable Length Keys. In *Proc. AFIPS Western Joint Computer Conference*, 1959.

[3] M. Erwig and M. Schneider. Formalization of Advanced Map Operations. In *9th Int. Symp. on Spatial Data Handling*, pages 8a.3–17, 2000.

[4] E. Fredkin. Trie Memory. *Communications of the ACM*, 3(9):490–499, 1960.

[5] M. McKenney. *Map Algebra: A Data Model and Implementation of Spatial Partitions for Use in Spatial Databases and Geographic Information Systems*. PhD thesis, University of Florida, August 2008.

[6] M. McKenney. Region extraction and verification for spatial and spatio-temporal databases. In *SSDBM*, Lecture Notes in Computer Science, pages 598–607. Springer, 2009.

[7] M. McKenney and M. Schneider. Topological Relationships Between Map Geometries. In *Advances in Databases: Concepts, Systems and Applications, 13th International Conference on Database Systems for Advanced Applications*, 2007.

[8] M. Schneider and T. Behr. Topological Relationships between Complex Spatial Objects. *ACM Trans. on Database Systems (TODS)*, 31(1):39–81, 2006.

[9] M. Scholl and A. Voisard. Thematic Map Modeling. In *SSD '90: Proceedings of the first symposium on Design and implementation of large spatial databases*, pages 167–190, New York, NY, USA, 1990. Springer-Verlag New York, Inc.

[10] C. D. Tomlin. *Geographic Information Systems and Cartographic Modelling*. Prentice-Hall, 1990.