In proceedings EGIS/MARI'94, Paris, March 29 - April 1, 1994

An R-tree based Map-Overlay Algorithm

Peter van Oosterom
TNO Physics and Electronics Laboratory,
P.O. Box 96864, 2509 JG The Hague, The Netherlands.
Email: oosterom@fel.tno.nl.

This paper presents a new vector map-overlay algorithm, which is based on a spatial access method: the R-tree. Combining and integrating spatial data from different sources in one of the key capabilities of a Geographic Information System. Usually the data, from these different sources, are stored in individual map layers and have to be combined by a map-overlay (or spatial join) operation. The new method is truly based on the "local processing" principle and is capable of dealing with non-uniform distributed data. In addition, it has the following advantages: is intuitively simple and attractive, produces an R-tree as side-product, also works for map-overlay with more than two input maps, and could be modified to work with polylines instead of line-segments as (internal computation) primitive. Performance tests confirm that the R-tree based map-overlay algorithm is very efficient.

1 Introduction

Most Geographic Information Systems (GISs) use map layers to organize the geographic features. Each layer describes a certain aspect of the modeled real world (de Hoop, van Oosterom, & Molenaar, 1993). This is the first reason for using map layers: it is a *natural* technique to organize the data from possibly different sources. The second reason for using map layers is that it can more *efficient* in terms of data storage and data manipulation. In the case that a lot of different thematic layers are stored together in one large structure map layer, the area and line features are chopped into many pieces (Bennis, David, Morize-Quilio, Thévenin, & Viémont, 1991; Bennis, David, Quilio, & Viémont, 1990), because topology rules require that there is a node at every intersection (Peucker & Chrisman, 1975; Boudriault, 1987).

Storing map layers separately, makes it impossible to directly solve topological queries that relate to features which belong to different layers. In order to solve queries that deal with multiple layers, a map-overlay operation has to be performed first. This makes map-overlay one of the key operations in a GIS as it allows the user to integrate spatial data.

In Section 2, four known algorithms for map-overlay are described together with their pro's and con's. Though several algorithms have been published, a new one is presented because the others have some serious drawbacks. The plane-sweep algorithms first sort the points of line-segments based on their x-coordinate (not a equal role for x and y) and could make better use of the "local processing" principle by taking both dimensions at the same time into account. This is what happens in the uniform grid algorithm, but this algorithm has another drawback: it does not perform well when data is non-uniform distributed.

Section 3 presents the new, R-tree based map-overlay algorithm which tries to overcome the problems of the existing algorithms. Topology reconstruction and attribute propagation are topics of the subsequent section. The worst case analysis, the implementation, the practical test results, and two further improvements are given in Section 5. Finally, the conclusions and future work are described in Section 6.

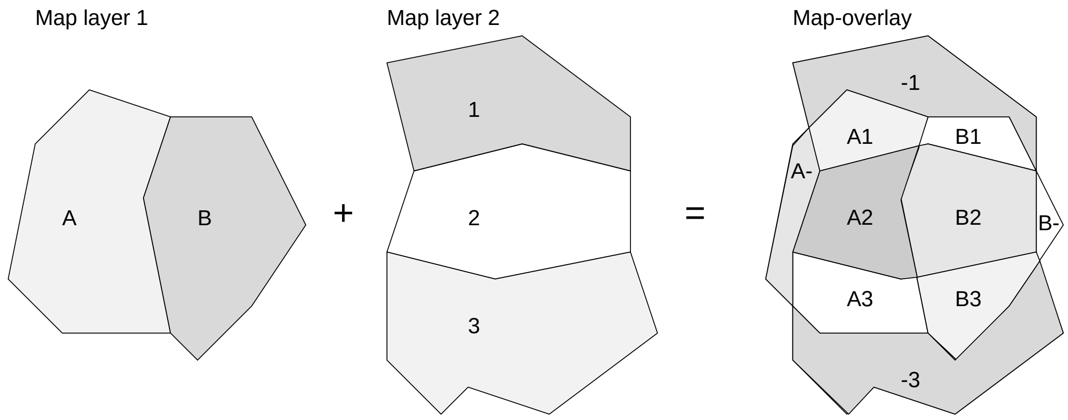


Fig. 1: Map-overlay computation and label propagation

2 Existing Map-Overlay Algorithms

The input of a map-overlay operation consists of two or more topologically structured layers. The output is a new layer in which the new areas are attributed based on the input layers; see Figure 1. Note that the spatial domains of the different input layers do not have to match exactly. The map-overlay is usually computed in three logical phases (Frank, 1987). The first step is performed at the metric level and computes all *intersections* between the edges (line segments) from the different layers. Followed by a reconstruction of the *topology* and assignment of labels or *attribute* values in the next two steps; see Section 4.

In this section, four alternatives for the first, and computationally most expensive, step of the map-overlay process are reviewed:

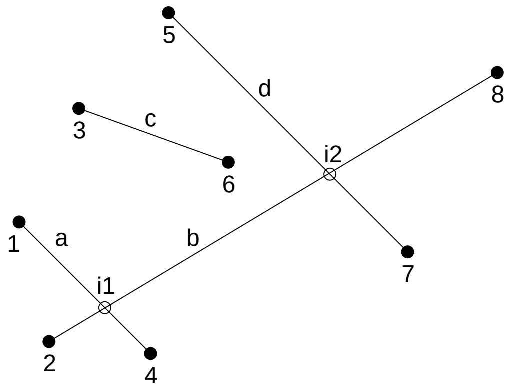
- brute force method;
- plane-sweep method (including several variants);
- uniform grid method;
- z-order-based method;

The *brute force* algorithm tests every line segment from one layer with every line segment from the other layer for intersection. Assuming that both map layers contain $O(n)$ line segments, then this process takes $O(n^2)$, which too much if one realizes that in practice n can be 100,000 or more.

More efficient algorithms try to take advantage of the "local processing" principle: line segments in one region do not interact with line segments in another (remote) region and should not be tested for intersection. Bentley and Ottman give one of the first descriptions of an application of this principle: the plane-sweep algorithm (1979) for reporting the intersections of line-segments. The *plane-sweep* algorithm first puts all the end points in a priority queue Q based on their x-coordinate. Later on, also the intersection points will be inserted into Q. Further, an initially empty list of "active" line segments (*status line* implemented by a balanced tree R) is ordered on y-coordinate. Now, points are removed from Q one by one. Every time a point p is removed one of the following tree events may happen:

1. p is a left end point of line segment s: s is inserted in R (based on y-coordinate) and if s intersects with the line segments immediately above or below, then insert intersection point(s) in Q;
2. p is a right end point of s: s is removed from R and if the segments immediately above and below s do intersect, then insert this intersection point in Q (if it is not already there);
3. p is an intersection point of segments s and t: swap s and t in R and when they have intersections with their new neighbors, insert this into Q.

The points are removed from the priority queue Q based on their x-coordinate starting with the lowest. In this way the status line advances from left to right, which explains the name of the algorithm: plane-



Priority queue Q	Event type	Status line R
1 2 3 4 5 6 7 8	1	-
2 3 4 5 6 7 8	1	a
3 i1 4 5 6 7 8	1	b a
i1 4 5 6 7 8	3	b a c
4 5 6 7 8	2	a b c
5 6 7 8	1	b c
6 7 8	2	b c d
i2 7 8	3	b d
7 8	2	d b
8	2	b
-		-

Fig. 2: The event queue Q and status line R during the plane-sweep

sweep. When k is the number of reported intersections, then $2n + k$ points have been removed from the queue Q. As each update on both Q and R takes $O(\log n)$, the algorithm runs in $O((n + k) \log n)$ time.

Nievergelt and Preparata (1982) give some refinements for polygonal maps: two planar convex subdivisions can be processed in $O(n \log n + k)$ time. Mairson and Stolfi (1988) showed that the same result can also be obtained for polygonal maps that need not to be convex. Quite recently, Chazelle and Edelsbrunner (1992) give an optimal $O(n \log n + k)$ algorithm that does not require the input data to be two planar subdivisions. Most algorithms require $O(n + k)$ space. A textbook description of the plane-sweep method can be found in (Preparata & Shamos, 1985). WHIRLPOOL (Dougenik, 1979; Chrisman, Dougenik, & White, 1992) is one of the first systems that applied the plane-sweep approach.

Kriegel et al. (1991*a*, 1991*b*) describe two manners to use a spatial access method in combination with plane-sweep to get performance improvements: The first method, called *sweep-line partition* selects input polygons from the spatial database as soon as they are near the sweep line. In this way the number of page faults is reduced by not loading all input polygons at the initial phase of the process. The other method, called *strip plane-sweep* decomposes the problem into horizontal strips which can be processed by a plane-sweep algorithm in the main memory of the computer. Special attention has to be given to make the processed chunks fit together.

Pullar describes another method based on the plane-sweep approach, called *X-ordering* (1990). This method does not work with line segments, but with more economic chains: connected line segments with monotonically increasing x-coordinate. First, all chains are sorted on their leftmost point. The chains are then processed one by one starting with the leftmost. The first chain is removed from the list, it becomes the current chain and it is checked for intersection with all other chains in the list until the leftmost point of the chains in the list is higher than the rightmost point of the current chain. Now the current chain is done. Empirical analysis (Pullar, 1990) shows that X-ordering out performs the normal plane-sweep method (for a data set with long chains). Drawback of this method is that there is not an equal role for x and y, just like the normal plane-sweep approach, and better use of the "local processing" principle cloud be made. So, chains that are far apart in y-direction are tested for intersection because they do overlap in the x-direction.

Franklin et al. (1989, 1989) suggest to use a uniform grid to organize the lines: the *uniform grid* method. This method does treat the x- and y-direction in an equal manner. The line intersection does only have to be performed for lines located in cells at the same location. This results in huge performance enhancements for the map-overlay operation. However, the method works optimal when the lines are distributed in a uniform manner. That is, every cell contains more or less the same number of lines. In the case that the distribution is not uniform, which is very usual for geographic data, the performance degrades because one cell contains a lot of lines which all have to be tested for intersection with a local quadratic time brute force algorithm.

Franklin (1990) also presents an efficient method, based on uniform grids, to compute the map-overlay polygons' areas without explicitly calculation the polygons. In other papers, Frankin (1987, 1988) describes how a Prolog implementation of map-overlay uses a grid to efficiently find intersecting edges. The detection of intersections using a uniform grid method can be implemented very well in a parallel computing environment. Care has to be taken that the second phase of the map-overlay (polygon reconstruction) is also implemented in a parallel manner (Waugh & Hopkins, 1992). Pullar combines the uniform grid method and the X-ordering method: *tiling scheme*. Longer grids (tiles) are used and instead of local brute force, the local X-ordering method is used.

Orenstein (1991) describes a map-overlay algorithm for k-dimensional (k not necessarily 2) situations based on the *z-order*; also called Morton key or Peano key. The z-order results from bitwise interleaving coordinates of a k-dimensional point. All objects are first approximated by boxes (which may be of different sizes: larger boxes are described by fewer bits) according to a space filling z-curve. These boxes are inserted into a one dimensional index structure sorted on the z-value. Map-overlay is now similar to merging sorted linear lists.

3 A New Map-Overlay Algorithm

Though the plane-sweep algorithm runs in the proven optimal theoretic time of $O(k + n \log n)$, in practice it might be out performed by other methods. This is caused by the fact that the plane-sweep algorithm first sorts on x-coordinate and that lines which are far apart (based on their y-coordinate) are active at the same time. For large data sets this means that active data set does not fit into main memory, which results in page faults and swapping with the associated delays. The uniform grid method adheres more closely to the local processing principle but is less suited for non-uniform distributed data.

This section presents a map-overlay algorithm based on the spatial index structure: the R-tree (Guttman, 1984). The basic operation in map-overlay is the intersection of two line segments. In principle, every line segment of the first map layer has to be tested for intersection with every line segment of the second map layer. As explained in the previous section, this would make map-overlay a very expensive operation, because the number of line segments in a map are usually very large.

In order to be able to deal with non-uniform distributed data, an R-tree is used. The insert algorithm of the R-tree will generate more nodes (cells) in areas with a lot of line segments. So the data structure automatically adapts itself to the distribution of the data. The pseudo-code in Figure 3 roughly outlines the kernel of the new map-overlay algorithm. The algorithm uses the following R-tree functions:

- `CreateRtree()`: creates a new R-tree;
- `DeleteRtree()`: deletes the R-tree;
- `InsertRtree(line)`: insert new entry (line segment);
- `RemoveRtree(line)`: removes entry;
- `RetrieveFirstRtree(box)`: initialize search box and returns first entry that overlaps or `NULL_ID` in case of no overlaps;
- `RetrieveNextRtree()`: returns next entry that overlaps the box set by `RetrieveFirstRtree` or `NULL_ID` when there are no more overlaps.

The basic principle of the algorithm is that everything stored in the R-tree is correct. That is, there are no intersections without nodes and three or more lines meeting in a node all have exactly the same coordinate at this node. This is required for the reconstruction of the

Fig. 3: The map-overlay algorithm

```
typedef line_segment              // data structure
   point  begin, end
   int    line_code, left_area_code, right_area_code
   int    map_nr

MapOverlay                        // main program
   CreateRtree();
   for "all line segments in map 1" do InsertRtree(line);
   for "all line segments in map 2" do CheckedInsert(line);
   // now there are no lines that intersect (without a nodes)
   FormAreaLoops();
   ComputeAttribtes();
   // Rtree can be very useful, but if not needed: DeleteRtree()

CheckedInsert(new_line)           // recursive function
   Intersected=FALSE;
   overlap_line=RetrieveFirstRtree(box_of(new_line, epsilon));
   while "overlap_edge != NULL_ID" and "not Intersected" do
      if Intersect(overlap_line, new_line, over1, over2, new1, new2) then
         Intersected=TRUE;
         RemoveRtree(overlap_line);
         InsertRtree(over1);       InsertRtree(over2);
         CheckedInsert(new1);      CheckedInsert(new2);
      else
         overlap_line=RetrieveNextRtree();
      end_if;
   end_while;
   if "not Intersected" then InsertRtree(new_line);
```

topology of the combined map layer. It is assumed the first map-layer is already correct. Therefore, all lines can be inserted into the R-tree without checking: `InsertRtree`.

The lines from the second map layer are inserted with the recursive function `CheckedInsert`. Note that this map-overlay algorithm can be extended any number of input maps by repeating this step for every map layer. Also, if the first map layer is not correct, then these lines should also be inserted with `CheckedInsert` instead of just `InsertRtree`. In `CheckedInsert` a new line is tested for overlapping bounding boxes within a distance *epsilon* with the earlier inserted lines in the R-tree: `RetrieveFirstRtree`. It is very important that the value of epsilon is set properly, because we deal with computers that have only finite precision floating-point arithmetic (Goldberg, 1991; Pullar, 1991).

In the case that there is an epsilon-overlap with an other line, the `Intersect` function has to determine if the lines really intersect or (epsilon) touch, or if they are disjoint and only the bounding boxes have overlap. In the later situation `Intersect` returns `FALSE` and the search in the R-tree for possible overlaps continues: `RetrieveNextRtree`. When the `new_line` does not intersect or touch any of the line segments in the R-tree, then it may savely be inserted with `InsertRtree`.

In the situation that `Intersect` really detects an intersect or (epsilon) touch with the `overlap_line`, then we must be careful not to chance the coordinates of its end points, because these must remain the same. Only the coordinates of the `new_line` may be snapped to the `overlap_line`. In this way a point is never moved more that the epsilon distance; solving thus the *creep* problem (Pullar, 1993). A lot of different relationships between two labeled line segments have to be considered in the `Intersect` function; see Figure 4.

In the case of a "normal" intersection (case 13; Figure 4), `Intersect` returns four new line segments: `over1` and `over2` are the result of splitting the `overlap_line` at the intersection; `new1` and `new2` have a similar relationship to `new_line`. The new line segments get the same attribute values (`line_code`,

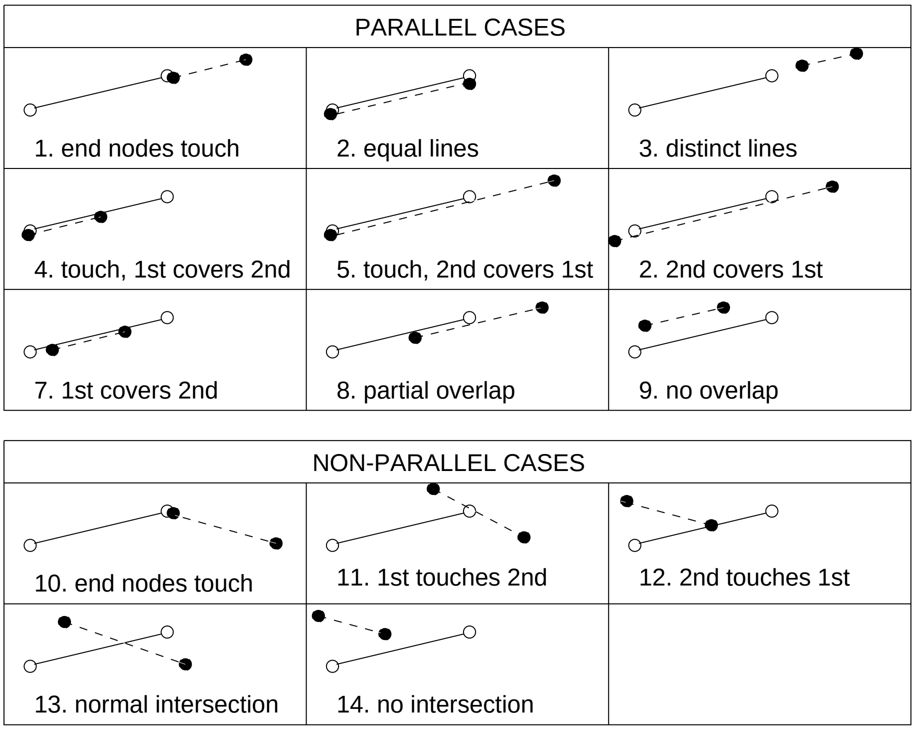


Fig. 4: The different relationships between two line segments (Note that the 1st line is drawn solid with open circles; 2nd line is drawn dashed with closed circles at the end points. Also the 2nd line is shown with a small offset in order to make the figure more clear)

`left_area_code, right_area_code`) as their "parent" line segment.

When the `overlap_line` has really been splitted, it is removed from the R-tree with `RemoveRtree`. The two new segments `over1` and `over2` can be savely inserted with `InsertRtree` without checking, because they are correct. This does not need to be true for the other two new segments `new1` and `new2`, so they are inserted with a recursive call to `CheckedInsert`. Note that the `Intersect` function does not always have to return four new line segments; e.g. touching line segments. However, these details are omitted from the pseudo code.

4 Topology reconstruction and attribute propagation

This section deals with the map-overlay operation aspects which occur after the computation of the intersections of the lines. It deals with the `FormAreaLoops` and `ComputeAttribtes` functions in Figure 3. Note that it might be practical to integrate these two steps. When all line segments of the first and the second map are inserted, the area loops can be formed. This process is not described in detail, only a global outline follows. All nodes have a sorted list of line segments by incoming angle; see Figure 5. Dangling nodes (have only one incoming edge) and dangling edges are removed. Finally, the loops are formed.

A loop is created by going from edge to next edge in the sorted edge list at the end node of the current edge. This process is repeated until the first edge is reached again. Special attention have to be paid to "island" loops, which can be detected by there negative orientation. Islands have to be linked in the topological data structure to their parent polygon. Finding the parent is based on spatial searching using the R-tree and a point-in-polygon test (Foley & van Dam, 1982). The whole loop creation process is terminated when every edge has been used twice: once left and once right.

The labeling (with two codes: map 1 and map 2) of the new area features, can be done through the original left/right codes of the

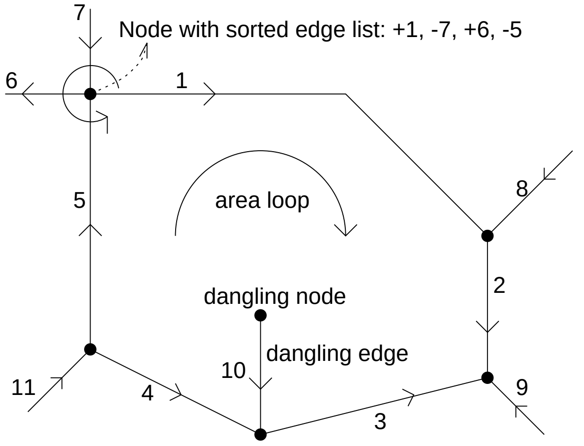


Fig. 5: Topological structure for nodes and dangling edge/node

line segments (van Roessel, 1990). This can only be done if the loop of an area feature contains line segments from both input map layers. If the loop contains just line segments from map 1, then the map 1 area code can be set directly and the map 2 area code must be the same as the map 2 area code of his neighbors. It is assumed that the map 2 code is already known in the neighbor. If this is not the case, then the final labeling of the area should be postponed to the "second round". Another possibility is to use a point from the border in a point-in-polygon test with the appropriate candidate polygons from map 2.

An alternative method is based on first determine a point inside the new area, a so called centroid. This is not as easy as one might expect. For example the center of gravity, which can be computed with a simple formula taking linear time (in the number of polygon points), may not fall inside an concave polygon. A good definition of the centroid might be the center of the largest circle that fits inside the polygon. This point can be found on the skeleton of the polygon. This skeleton can be created by first building the line-Voronoi diagram, which can be achieved efficiently in $O(n \log n)$ (Chew & Kedem, 1989; Aonuma, Imai, Imai, & Tokuyama, 1990; Preparata & Shamos, 1985).

By inspection, the location of the centroid in maps 1 and 2 (using the well known point-in-polygon test), the codes for labeling can be obtained. This can be done efficiently is there is spatial search structure (e.g. an R-tree) for the input maps.

As mentioned in the previous section, the R-tree based map-overlay method can easily be extended to overlay more than two map layers. The geometric computations remain the same, only the labeling has to be adapted. Labeling is related to the issue of *attribute propagation*, instead of labeling the new areas with codes users are more interested in the attributes. The thematic attributes of the original map layers can be:

- all propagated to the new layer;
- partially propagated, that is a selection of attribtes are associated with the new layer;
- used in functions to compute the new thematic attributes;

Van Roessel (1990) discusses attribute propagation in depth and describes an efficient method for efficiently propagating attributes in the situation of plane-sweep overlay algorithm. An other problem related to map-overlay is the introduction of sliver polygons. Solutions for this have been presented in (Chrisman, 1983; Zhang & Tulip, 1990).

A formal algebraic specifications of the (raster) map-overlay operation, in which the non-spatial attributes are combined, has been described in (Dorenbeck & Egenhofer, 1991). Using this formalism makes it possible to find more efficient processing strategies for several overlay operations.

5 Performance evaluation

In this section, a worst case analysis of the R-tree based map-overlay algorithm is given, followed by a short description of the implementation, test results, and two suggestions for further improvements. Map-overlay with n input line segments and k intersections, requires at most k R-tree entry deletes and $n + 3k$ inserts. Each delete or insert takes about $O(\log n)$ as the R-tree is balanced. Therefore, the total worst case time required by the map-overlay operation is $O((n+k)\log(n+k))$ and the required space is $O(n+k)$.

Some performance tests indicate that the maintenance of the R-tree is the current bottleneck in our implementation. Therefore, the R-tree must be tuned for fast creation[1]. Calculating a map-overlay of a map with about 30.000 line segments with its own mirrored counterpart takes about 14 min CPU time on a Sun Sparctation 2 with 32 Mb main memory. The resulting map-overlay contains a little over 100,000 line segments. We should compare our map-overlay algorithm to other algorithms by implementing these algorithms in the same environment. Unfortunately, this has not yet been done.

Two possible improvements to the presented algorithm are: 1. use polylines instead of line segments and 2. try to *merge* existing R-trees which correspond to the input map layers. The advantages of using polylines are clear: memory overhead is reduced (line segments duplicate the coordinates) and topology reconstruction becomes easier because segments are already connected. The drawback is that the `Intersect` function becomes more complicated and can return any number of intersected polyline parts.

Merging R-trees might be difficult, perhaps even impossible. However, if the lines from the second map are inserted into the overlay in the order based on an index scan of the R-tree (of the second map), then this might be enough. Using the index scan will insert the new lines in a spatially coherent way. This means that the same parts of the map-overlay R-tree are concerned and that page faults are reduced.

6 Conclusion and Future Work

In this paper a new R-tree based map-overlay algorithm has been presented. It is possible to base this map-overlay algorithm on other spatial access methods: quadtree (Samet, 1989), R*-tree (Beckmann, Kriegel, Schneider, & Seeger, 1990), KD2B-tree, Sphere-tree (van Oosterom & Claassen, 1990), Cell-tree (Günther, 1989), etc. Besides its good performance and simplicity, this approach also produces a spatial index of the resulting new map layer.

The test results of the implementation indicate that the performance is very good. However, more comparative tests will have to be executed to compare the new method with the existing methods: plane-sweep (Bentley & Ottmann, 1979), uniform grid (Franklin *et al.*, 1989), and z-order based approaches (Orenstein, 1991). The presented R-tree based map-overlay algorithm is available in the GEO++ system (van Oosterom & Vijlbrief, 1991; Vijlbrief & van Oosterom, 1992) as one of the standard spatial analyses tools.

[1] Some practical guidelines for tuning the R-tree: set the maximum number of entries per node not to high (between 10 and 20) and set minimum number of entries very low; e.g. 2.

References

Akman, V., Franklin, W. R., Kankanhalli, M., & Narayanaswami, C. (1989). Geometric computing and uniform grid technique. *Computer Aided Design*, *21(7)*, 410–420.

Aonuma, H., Imai, H., Imai, K., & Tokuyama, T. (1990). Maximin Location of Convex Objects in a Polygon and Related Dynamic Voronoi Diagrams. in *Proceedings of the 6th ACM Symposium on Computational Geometry*, pp. 225–234.

Beckmann, N., Kriegel, H.-P., Schneider, R., & Seeger, B. (1990). The R^*-tree: An Efficient and Robust Access Method for Points and Rectangles. in *ACM/SIGMOD, Atlantic City*, pp. 322–331 New York. ACM.

Bennis, K., David, B., Quilio, I., & Viémont, Y. (1990). GéoTropics Database Support Alternatives for Geographic Applications. in *4th International Symposium on Spatial Data Handling, Zürich*, pp. 599–610 Columbus, OH. International Geographical Union IGU.

Bennis, K., David, B., Morize-Quilio, I., Thévenin, J. M., & Viémont, Y. (1991). GéoGraph: A Topological Storage Model for Extensible GIS. in *Auto-Carto 10*, pp. 349–367.

Bentley, J. L., & Ottmann, T. A. (1979). Algorithms for Reporting and Counting Geometric Intersections. *IEEE Transac-*

tions on Computers, *C-28(9)*, 643–647.

Boudriault, G. (1987). Topology in the TIGER File. in *Auto-Carto 8*, pp. 258–269.

Chazelle, B., & Edelsbrunner, H. (1992). An Optimal Algorithm for Intersecting Line Segments in the Plane. *Journal of the Association for Computing Machinery*, *39(1)*, 1–54.

Chew, L. P., & Kedem, K. (1989). Placing the Largest Similar Copy of a Convex Polygon Among Polygonal Obstacles. in *Proceedings of the 5th ACM Symposium on Computational Geometry*, pp. 167–174.

Chrisman, N. R., Dougenik, J. A., & White, D. (1992). Lessons for the Design of Polygon Overlay Processing from the ODYSSEY WHIRLPOOL Algorithm. in *Proceedings of the 5th International Symposium on Spatial Data Handling, Charleston, South Carolina*, pp. 401–410 Columbus, OH. International Geographical Union IGU.

Chrisman, N. R. (1983). Epsilon filtering – A technique for automated Scale Changing. in *Technical Papers of the 43rd Annual Meeting of the American Congress on Surveying and Mapping*, pp. 322–331.

de Hoop, S., van Oosterom, P., & Molenaar, M. (1993). Topological Querying of Multiple Map Layers. in *COSIT'93, Elba Island, Italy*, pp. 139–157 Berlin. Springer-Verlag.

Dorenbeck, C., & Egenhofer, M. J. (1991). Algebraic Optimization of Combined Overlay Operations. in *Auto-Carto 10*, pp. 296–312.

Dougenik, J. (1979). WHIRLPOOL: A Geometric Processor for Polygon Coverage Data. in *Auto-Carto 4*, pp. 304–311.

Foley, J. D., & van Dam, A. (1982). *Fundamentals of Interactive Computer Graphics*. Addison-Wesley, Reading, Mass.

Frank, A. U. (1987). Overlay Processing in Spatial Information Systems. in *Auto-Carto 8*, pp. 12–31.

Franklin, W. R., & Wu, P. Y. F. (1987). A Polygon Overlay System in PROLOG. in *Auto-Carto 8*, pp. 97–106.

Franklin, W. R., Narayanaswami, C., ans David Sun, M. K., Zhou, M.-C., & Wu, P. Y. P. (1989). Uniform Grids: A Technique for Intersection Detection on Serial and Parallel Machines. in *Auto-Carto 9*, pp. 100–109.

Franklin, W. R. (1990). Calculating Map Overlay Polygons' Areas without Explicitly Calculating the Polygons: Implementation. in *4th International Symposium on Spatial Data Handling, Zürich*, pp. 151–160 Columbus, OH. International Geographical Union IGU.

Goldberg, D. (1991). What Every Computer Scientist Should Know About Floating-point Arithmetic. *ACM Computing Surveys*, *23(1)*, 5–48.

Günther, O. (1989). The Design of the Cell Tree: An Object-Oriented Index Structure for Geometric Databases. in *Proceedings IEEE Fifth International Conference on Data Engineering, Los Angeles, California*, pp. 598–605 Los Alamitos, CA. IEEE Computer Society Press.

Guttman, A. (1984). R-Trees: A Dynamic Index Structure for Spatial Searching. *ACM SIGMOD*, *13*, 47–57.

Kriegel, H.-P., Brinkhoff, T., & Schneider, R. (1991*a*). The Combination of Spatial Access Methods and Computational Geometry in Geographic Database Systems. in *Advances in Spatial Databases, 2nd Symposium, SSD'91, Zürich*, pp. 3–21 Berlin. Springer-Verlag.

Kriegel, H.-P., Brinkhoff, T., & Schneider, R. (1991*b*). An Efficient Map Overlay Algorithm Based on Spatial Access Methods and Computational Geometry. in *Geographic Database Management Systems*, pp. 194–211 Berlin. Springer-Verlag.

Mairson, H. G., & Stolfi, J. (1988). Reporting and Counting Intersections between Two Sets of Line Segments. in *Theoretical Foundations of Computer Graphics and CAD*, Vol. 40 of *NATO ASI Series*

F, pp. 307–325 Berlin. Springer-Verlag.

Nievergelt, J., & Preparata, F. P. (1982). Plane-Sweep Algorithms for Intersecting Geometric Figures. *Communications of the ACM*, *25(10)*, 739–747.

Orenstein, J. (1991). An Algorithm for Computing the Overlay of k-Dimensional Spaces. in *Advances in Spatial Databases, 2nd Symposium, SSD'91, Zürich*, pp. 381–400 Berlin. Springer-Verlag.

Peucker, T. K., & Chrisman, N. (1975). Cartographic Data Structures. *American Cartographer*, *2(1)*, 55–69.

Preparata, F. P., & Shamos, M. I. (1985). *Computational Geometry*. Springer-Verlag, New York.

Pullar, D. (1990). Comparative Study of Algorithms for Reporting Geometrical Intersections. in *4th International Symposium on Spatial Data Handling, Zürich*, pp. 66–76 Columbus, OH. International Geographical Union IGU.

Pullar, D. (1991). Spatial Overlay with Inexact Numerical Data. in *Auto-Carto 10*, pp. 313–329.

Pullar, D. (1993). Consequences of Using a Tolerance Paradigm in Spatial Overlay. in *Auto-Carto 11*, pp. 288–296.

Samet, H. (1989). *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, Reading, Mass.

van Oosterom, P., & Claassen, E. (1990). Orientation Insensitive Indexing Methods for Geometric Objects. in *4th International Symposium on Spatial Data Handling, Zürich*, pp. 1016–1029 Columbus, OH. International Geographical Union IGU.

van Oosterom, P., & Vijlbrief, T. (1991). Building a GIS on Top of the Open DBMS "Postgres". in *Proceedings EGIS'91: Second European Conference on Geographical Information Systems*, pp. 775–787 Utrecht. EGIS Foundation.

van Roessel, J. W. (1990). Attribute Propagation and Line Segment Classification in Plane-Sweep Overlay. in *4th International Symposium on Spatial Data Handling, Zürich*, pp. 127–140 Columbus, OH. International Geographical Union IGU.

Vijlbrief, T., & van Oosterom, P. (1992). The GEO System: An Extensible GIS. in *Proceedings of the 5th International Symposium on Spatial Data Handling, Charleston, South Carolina*, pp. 40–50 Columbus, OH. International Geographical Union IGU.

Waugh, T. C., & Hopkins, S. (1992). An algorithm for polygon overlay using cooperative parallel processing. *International Journal of Geographical Information Systems*, *6(6)*, 457–467.

Wu, P. Y. F., & Franklin, W. R. (1988). A logic Programming Approach To Cartographic Map Overlay. in *International Computer Science Conference, Hong Kong*.

Zhang, G., & Tulip, J. (1990). An Algorithm for the Avoidance of Sliver Polygons and Clusters of Points in Spatial Overlays. in *4th International Symposium on Spatial Data Handling, Zürich*, pp. 141–150 Columbus, OH. International Geographical Union IGU.