Published in: G. Gambosi, M. Scholl, H.-W. Six (eds.), “Geographic Database Management Systems”, Workshop Proceedings, Capri, May 1991, Springer-Verlag, 194-211, 1992.

An Efficient Map Overlay Algorithm Based on Spatial Access Methods and Computational Geometry[*]

Hans-Peter Kriegel, Thomas Brinkhoff, Ralf Schneider

Abstract

Geographic database systems, known as geographic information systems (GISs) particularly among non-computer scientists, are one of the most important applications of the very active research area named spatial database systems. Additionally to other spatial database systems where query and manipulation processing are emphasized, the most important purpose of a GIS is to analyze geographical data. The basic building block for analysis operations in GIS is the operation map overlay. However, available map overlay algorithms suffer from poor performance. In this paper, we present an efficient map overlay algorithm using as ingredients spatial access methods and state-of-the-art computational geometry concepts. An experimental performance analysis of our implemented map overlay algorithm demonstrates the fruitfulness of the marriage of spatial access methods with computational geometry methods for improving the efficiency of GISs. The possibility of coupling these two areas is based on the fact that both use spatial order relations in order to increase the performance of query processing.

Keywords: map overlay, computational geometry, spatial access methods

1 Introduction

Geographic information systems are one of the most important applications of spatial database systems. Basically, they consist of two parts: First, components to query and manipulate geographical data and second, components to manage and store the data. However, the main purpose of a GIS is to analyze geographical data. Map overlay is one of the most important analysis operations in a GIS, e.g. C.D. Tomlin's map analysis package (MAP) [Tom9Oj is completely based on the map overlay operation. The map overlay combines two or more maps of different topics into a single new map. The goals are to derive new maps, to find correlations between the information encoded in maps, and to process complex queries.

The algorithms presented in the past (e.g. in [Whi 78], [Fra 83], [FW 87]) assume that the input maps of the map overlay are kept in main memory or in sequential files on secondary storage. The following two important requirements of future GISs demand for new approaches: First, the database system of a GIS must be able to manage very large volumes of data. The large amount of data (in the order of Giga- and Terabytes) is additionally increased by pursuing the goal to manage scaleless and seamless databases [Oos 90]. Second, the database system has to support spatial access to parts of the database, such as maps, and to the objects of a map. Such access is a necessary condition for efficient query and manipulation processing.

Pursuing these goals, we want to take advantage of spatial access methods (SAMs). In the past few years, many access methods were developed which allow to organize large sets of spatial objects on secondary storage. There are three basic techniques, which extend multidimensional point access methods (PAMs) to multidimensional spatial access methods [SK 88]: clipping, overlapping regions, and transformation.

[*] This work was supported by grant no. Kr 670/4-3 from the Deutsche Forschungsgemeinschaft (German Research Society) and by the Ministry of Environmental and Urban Planning of Bremen.

Point access methods such as the grid file [NHS 84], PLOP-hashing [KS 88], the BANG file [Fre 87], and the buddy tree [SK 90] can be extended by these techniques. Additionally, there are access methods, which are designed for managing simple spatial objects directly. They use one of the above techniques inherently, e.g. the R-tree [Gut 84] and the R*~tree [BKSS 90] use overlapping regions, or the cell tree [Gün 89] uses clipping.

The use of SAMs as an ingredient in GISs is absolutely necessary to guarantee good retrieval and manipulation performance, in particular for large maps. The use of SAMs enables us to overlay only relevant parts of the seamless database. Additionally, we have the possibility to partition the maps and to combine the results of these partitions in order to increase the performance of the overlay. In addition to using access methods, we increase the performance of the map overlay operation in our approach applying a plane-sweep algorithm. Thus, we combine spatial access methods and computational geometry in order to improve the efficiency of GISs. The combination of these two areas is based on the fact that both use spatial order relations. To our knowledge this combination, although self-suggesting, has never been performed before.

The next section explains and formally defines the terms 'thematic map' and 'map overlay'. Our plane-sweep overlay algorithm is presented in section 3. An analytical and experimental performance analysis of the overlay operation is presented in section 4. The combination of the algorithm and spatial access methods follows in section 5. The paper concludes with a summary and an outlook to future work.

2 Definitions

2.1 Thematic maps

In this paper, the term map is used for *thematic maps*. Those emphasize one or more selected topics, e.g. land utilization, population density etc. Thematic maps are generally represented by *choropleth maps*, which separate areas of different properties by boundaries [Bur 86], e.g. forests, lakes, roads, or agriculturally used areas (see figure 1).

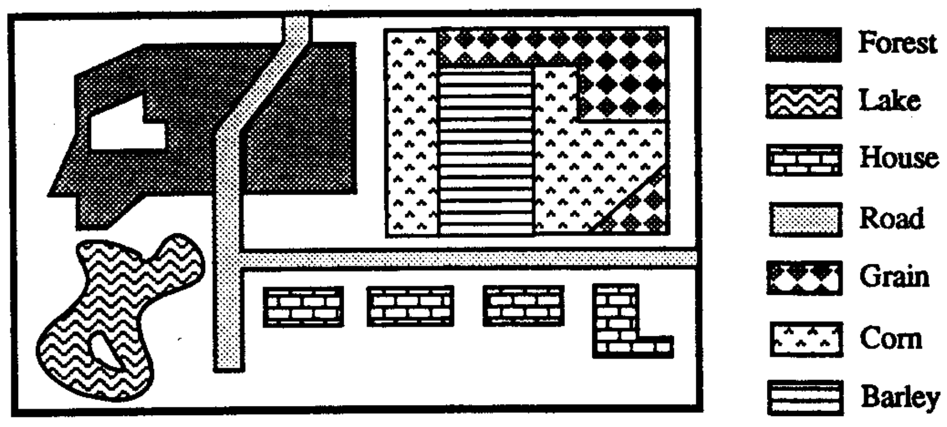

Figure 1: Example of a thematic map

We consider in this paper only maps modeled by a *vector representation* because there are two main disadvantages of raster representations [Oos 90]: (1) Raster data depends on a specific projection. Therefore, there are problems when combining raster maps from different sources. A scaleless database cannot be realized using a raster representation. (2) Objects in raster maps generally are not handled individually. Thus, a support by access methods is more difficult. Additionally, raster data are more voluminous.

We assume that the connected areas with the same property are described by *simple polygons with holes*, and that the used data structures are able to handle such polygons explicitly [KHHSS 91b]. A polygon is simple if there is no pair of non-consecutive edges sharing a point. A simple polygon with holes is a simple polygon where simple polygonal holes may be cut out (see figure 2). There may be other areas in such a hole. The areas of a map are disjoint but they do not need to cover the map completely. Each area refers to exactly one *thematic attribute*. In figure 1, these characteristics of a thematic map are depicted by an example, which visualizes the land utilization of a part of a map.

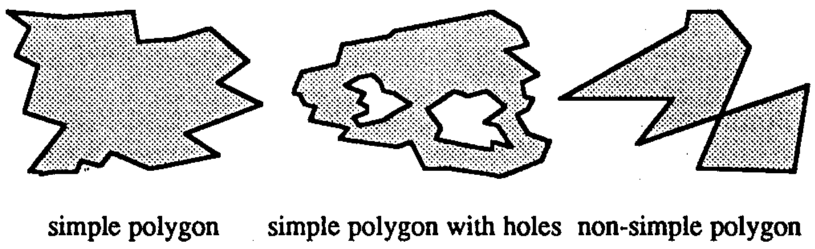

Figure 2: Different polygons

In the following a formal definition of thematic maps is presented where c~ denotes the regularized intersection [Til 80] and T is the set of values of the thematic attributes of the map M:

$M := \{ t = (t.P, t.A) \mid t.P$ is a simple polygon with holes, $t.A \in T \}$,
where $t_1, t_2 \in M, t_1 \neq t_2 \quad t_1.P \cap_r t_2.P = ø$

Maps of different topics describing the same part of the world are called *map layers*. The map overlay operation combines such layers.

2.2 Map overlay

The *map overlay* combines two or more maps (input maps) of different topics into a single new map (output map). An established definition of the map overlay operation on vector representation does not exist in the literature. The power of the algorithms presented in literature is not comparable (see [Bar 86]). Therefore it was necessary in our opinion to develop a formal definition of the map overlay which generalizes all these proposals and which is suitable for many purposes, e.g. all those claimed in MAP [Tom 90]. Our definition is based on the above specification of thematic maps. One important property of our definition is that parts of the map, which are not covered by each input map, may occur in the output map. The combination of the thematic attributes or of geometric or topological properties of the input areas is controlled by an *overlay function* f, where f is defined or selected by the user of the 015.

Before we present the formal definition of the map overlay we want to illustrate this operation by an example. Figure 3 depicts two input maps: the map 'land utilization' from figure 1 and another map 'soil pollution'. In the output map all areas should be reported, which are forests or agriculturally used land and where the degree of soil pollution is greater than 2.

In the following we present the formal definition of map overlay where M* denotes the set of all thematic maps. Furthermore, we introduce for each input map r_i to process the empty parts of the map correctly. r_i consists of r_i. P which represents the non-covered area of the map and where $r_i.A \notin T_i$. By not combining different r_i's, we prevent generating areas in the output map, which are not covered by any area. For this we use the intersection operation $\cap$', which is the same as the regularized intersection except for $r_i.P \cap_r r_j.P = ø$ To eliminate arena, which should not appear in the output map, we introduce an attribute r indicating an unwanted tuple. The set of connected but not expandable parts of A is denoted sep (A).

$$f(t_1, t_2) = \begin{cases} ▨ & (t_1.A = \text{grain or corn or barley}) \text{ and } (t_2.A \geq 2) \\ ▩ & (t_1.A = \text{forest}) \text{ and } (t_2.A \geq 2) \\ r & \text{otherwise} \end{cases}$$

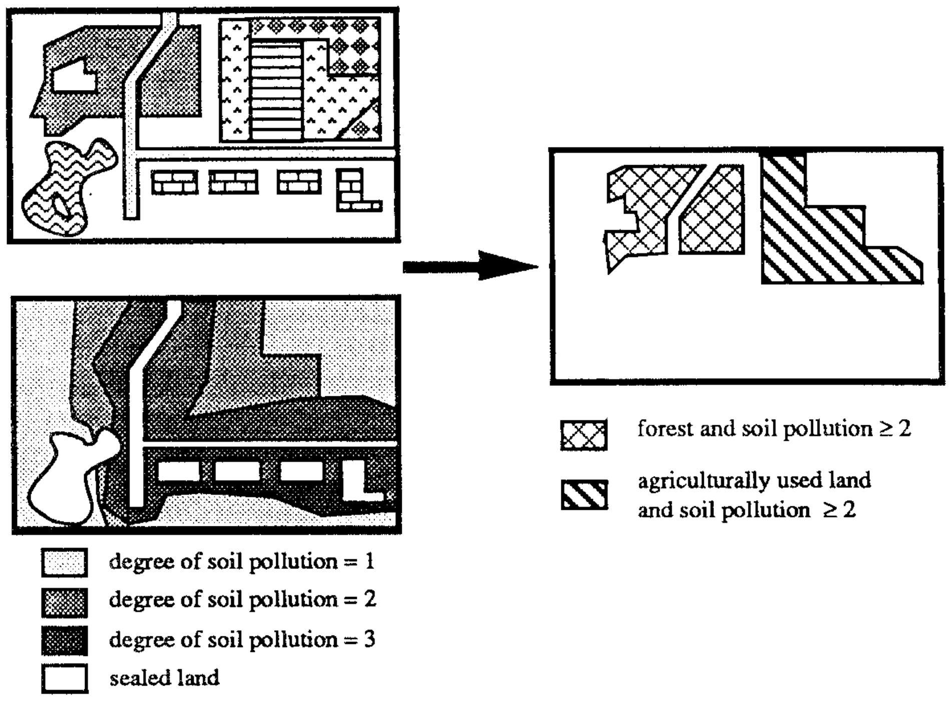
degree of soil pollution = 1
degree of soil pollution = 2
degree of soil pollution = 3
sealed land
forest and soil pollution ≥ 2
agriculturally used land and soil pollution ≥ 2

Figure 3: Example of a map overlay and an overlay function

$$\begin{aligned} \text{overlay: } & M^* \times M^* \times \{f \mid f: (M_1 \cup \{r_1\}) \times (M_2 \cup \{r_2\}) \rightarrow (T_3 \cup \{r\})\} \rightarrow M^* \\ & (M_1, M_2, f) \rightarrow M_3 \, , \\ \text{where } & M_3 = \{\, t = (t.P, t.A) \mid t.P \in \text{sep}\,(t_1.P \cap' t_2.P) \, , \, t.P \neq \emptyset \, , \\ & \qquad t.A = f\,(t_1, t_2),\ t.A \neq r,\ t_1 \in M_1 \cup \{r_1\} \, , \, t_2 \in M_2 \cup \{r_2\} \,\} \end{aligned}$$

Due to clarity, we restricted the definition to an overlay with two maps and an overlay function, which processes only tuples of the maps. The generalization of the definition for overlaying more than two maps is straightforward. For an integration of topological properties, e.g. neighborhood, only the parameter list of the overlay function f must be extended.

3 The plane-sweep overlay

A desirable overlay algorithm should define and utilize an order relation on the objects in the plane to enable a spatial partition of the input maps. Plane sweep is a technique of computational geometry, which fulfills this demand [PS 88]: the coordinates of objects *(event points)* are projected onto the x-axis and are processed according to the order relation on this axis. Event points are stored in a queue called *event point scheduler*. If event points are computed during processing, the event point scheduler must be able to insert event points after initialization. A vertical line sweeps the plane according to the event points from left to right. This line is called *sweep line*. The state of the plane at the sweep line position is recorded in a table called *sweep line status*. Event points, which are passed by the sweep line, are deleted from the event point scheduler. Figure 4 depicts an example of the event point scheduler and the sweep line status.

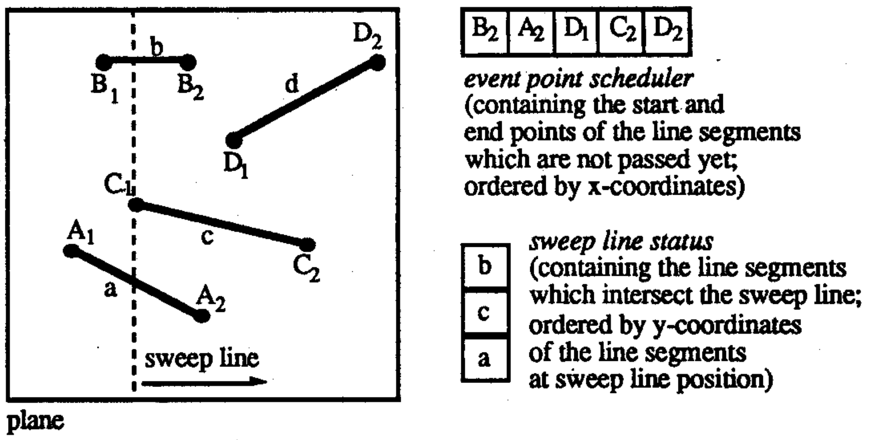
b
B1
B2
D2
d
D1
C1
c
A1
a
A2
C2
sweep line
plane
B2 A2 D1 C2 D2
event point scheduler
(containing the start and
end points of the line segments
which are not passed yet;
ordered by x-coordinates)
b
c
a
sweep line status
(containing the line segments
which intersect the sweep line;
ordered by y-coordinates
of the line segments
at sweep line position)

Figure 4: Example of a plane sweep

In order to simplify the following algorithms we assume that different event points have different x-coordinates. This assumption can be cancelled by introducing a more elaborate order criteria (e.g. event points with equal x-coordinates are sorted by their y- coordinates) and by treating vertical edges according to their direction.

3.1 Nievergelt and Preparata's algorithm to determine the regions of a self-intersecting polygon

Nievergelt and Preparata's algorithm [NP 82] is a plane-sweep algorithm to determine the regions of a self-intersecting polygon (see figure 5). It is based on an algorithm developed by Bentley and Ottmann [BO 79], which computes the intersecting points of a set of line segments. In Nievergelt and Preparata's algorithm the vertices and computed intersection points are stored in the event point scheduler. The edges intersecting the sweep line are kept in the sweep line status determined by their y-coordinate at the sweep line position. The event points are classified into one of four categories:

1. start point:	both edges at the event point are to the right of the point
2. end point:	both edges at the event point are to the left of the point
3. bend:	one edge is on the right and one is to the left of the event point
4. intersection point:	two edges intersect in the event point

For computing the regions between the edges, the sweep line status consists additionally of two pointers to point lists for each edge. The point lists describe the boundaries of the regions. A corresponding list is extended by an event point when the sweep line reaches this point depending on its category. For every region between two edges in the sweep line status there is an identification number (region ID). Figure 5 depicts an example.

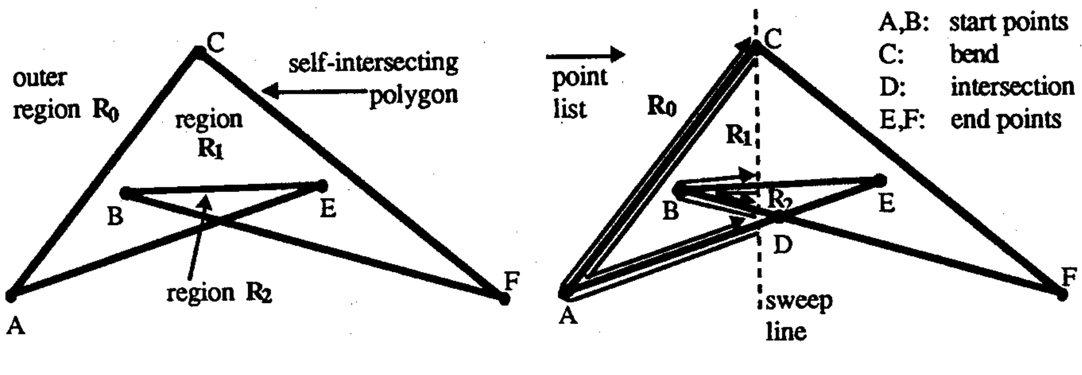

Figure 5: Example of Nievergelt and Preparata's algorithm [NP 82j

3.2 Our algorithm for map overlay

In [NP 82] Nievergelt and Preparata mentioned to generalize the classification of the event points for generating an algorithm, which intersects arbitrary maps. In the following we present and examine an algorithm for map overlay in detail, which uses the algorithm described above as a building block and which realizes the generalization of this classification. The regions evaluated by the algorithm form the areas of the output map. Obviously, not only the determination of regions is needed for an overlay, but also the algorithm has to identify the polygons of the input maps covering the new region. These polygons are called *parent polygons*. This step must be done to calculate the value of the thematic attribute of the region by the overlay function.

For handling the point lists correctly, every edge touching an event point must be attached to this point. There are edges starting, ending, or crossing the event point. We consider crossing edges as a combination of starting and ending edges (see figure 6). Therefore, the treatment of event points is processed in two steps:

1. Treat the edges to the left of the event point.
2. Treat the edges to the right of the event point.

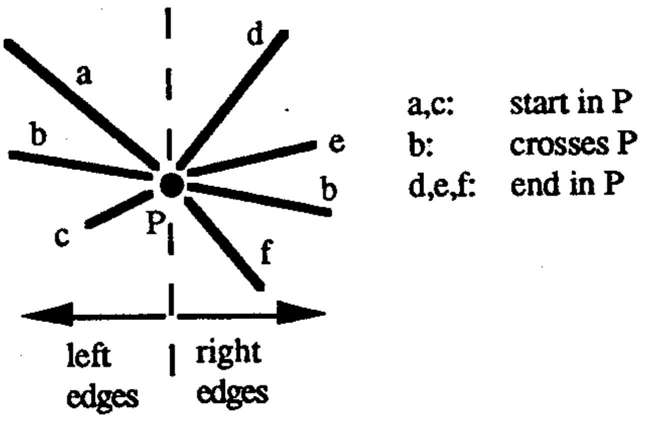

Figure 6: Example of an event point

As long as only regions of one polygon are computed, the classification of the event points is directly possible when the point is inserted into the event point scheduler. If more than one polygon is treated, the vertices or intersection points of different polygons may coincide. Such points must be combined in one event point. The edges attached to one event point are sorted by their gradient.

Two types of point lists describing regions can be distinguished: *Outer point lists* describe for each region the area outside of this region. Outer point lists are directed clockwise. Point lists describing the inner area of a region are called *inner point lists*. They are directed counterclockwise. The inner point list of a region is the outer one of a neighboring region separated by this list, and vice versa.

3.2.1 Treatment of the event points

Let P be the actual event point. Then s_l to s_n (t_l to t_m) are the left (right) edges attached to P ordered from top to bottom. For each edge k exist two pointers A(k) and B(k). A(k) points to the tail of the point list above the edge k and B(k) points to the head of the point list below the edge k. The sweep line status is denoted by y.

Initially, we inspect the edges on the left side (see figure 7). According to their number we proceed:

Case l: n = 0

A new outer point list is necessary which only consists of the point P. A(t_l) and B(t_m) must point to this list.

Case 2: n = 1

The two point lists referenced by A(s_l) and B(s_l) are extended by the point P and are attached to the pointers A(t_l) and B(t_m), respectively. s_l is deleted from y.

Case 3: n > 1

If m = 0, the outer point list must be closed. There exist two sub cases: If A(s_l) and B(s_n) point to the same list, this list (extended by P) describes a polygonal hole (or an outer region) which is stored in the output map. If A(s_l) and B(s_n) point to different lists, the lists and P are only concatenated.

If m> 0, the outer point lists A(s_l) and B(s_n) are extended by P and are attached to A(t_l) and B(t_m) as in case 2.

For all regions between the edges s_i and s_{i+1} exist also two sub cases: If B(s_i)=A(s_{i+1}), the inner list B(s_i) (extended by P) describes a new region which is stored in the output map. Otherwise A(s_{i+1}), P, and B(s_i) are concatenated.

s_l, …, s_n are deleted from y.

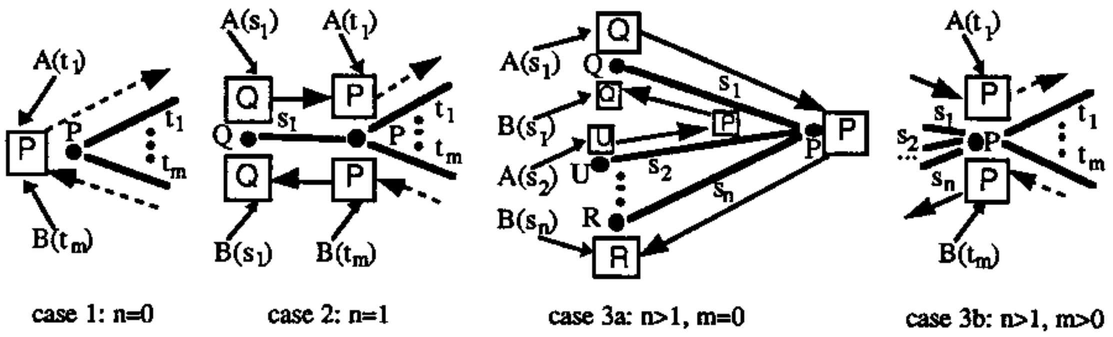

Figure 7: Treatment of left edges

The edges on the right side of the event point are examined in the second step. Additionally, intersection points are determined in this step. A computed intersection point is inserted into the event point scheduler. There are also three cases in the second step:

Case 1: m = 0

The two regions, which are separated by the edges s_l and s_n up to, now are identical (see figure 7, case 3a). Therefore, the region IDs must be adjusted if they are different. Because s_l to s_n are deleted from the sweep line status y, two edges of y, which were not neighbors, now are new neighbors and must be tested for intersection.

Case 2: m = 1

t_l is inserted into y and tested for intersection with its new neighbors.

Case 3: m > 1

t_l to t_m are inserted into y. Then t_l is tested for intersection with its new neighbor above and t_m is tested with its new neighbor below. For all i = 1, ... , m-l a new region is starting between the edges t_i and t_{i+1} and receives its own ID. Therefore, a new inner point list consisting of P is generated where the two pointers $A(t_i)$ and $B(t_{i+1})$ refer to this list (see figure 8).

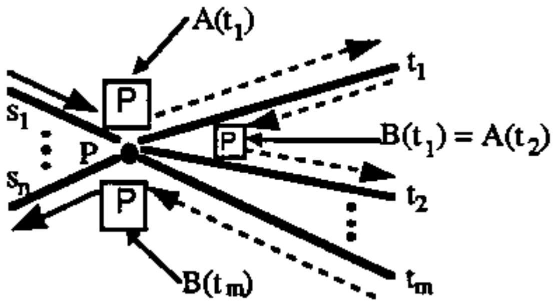

Figure 8: Treatment of right edges

A region, which describes a polygon hole, can be assigned to the surrounding region by the region IDs. A formal description of the algorithm is presented in the appendix.

3.2.2 Identification of the parent polygons

In the overlay operation, it is necessary to identify the parent polygons of a region because we need these polygons to determine the thematic attribute of each area in the output map evaluating the overlay function. Other overlay algorithms (e.g. [Fra 83]) are inefficient with respect to determining the parent polygons, e.g. costly point location tests have to be performed. Using the plane-sweep technique, our algorithm allows a simple and efficient solution because the neighborhood of the regions intersecting the sweep line is inherently stored by the sweep line status:

In every input map M_i, there exists exactly one or no parent polygon for every evaluated region. Therefore, we attach to every edge of an input polygon a lower and an upper mark initially. The mark, which is set, indicates the side where the polygon is located and contains a pointer to the polygon. The other mark is empty. While combining event points, coinciding edges are discovered. In this case, also the marks are combined. The set of parent polygons P(R) of a new region R is identified when the region is starting (step 2, case 3). At this point of time, the parent polygons of the region Q lying above R are known and also the marks of the edge e between R and Q:

$$P(R) = (P(Q) \setminus \text{upper mark } (e)) \cup \text{lower mark } (e)$$

This approach works also for holes by using exchanged marks.

3.3 A suitable coordinate representation for map overlay

The instability of plane-sweep algorithms against numerical errors is an objection being raised. This reproach may be justified if a floating-point representation is used to compute the intersection points. However, *rational coordinates* are a more suitable representation because they form a vector space. For example, a rational representation of coordinates is used in an implementation of a map overlay in [FW 87].

A more detailed analysis of a representation by rational coordinates leads to the following statements:

1. The coordinates in maps recorded by a GIS can be represented by pairs of integers. This assumption is realistic because both, the described part of the world and the resolution are limited.

2. To compute intersection points, integer coordinates are insufficient [Fra 84]. But a map overlay with input maps containing integer coordinates only, needs a limited number of digits to represent the intersection points by rational numbers. Let n the number of digits of the integers of the input map then the number of digits of the nominator of the intersection points is smaller than 2*n+4 and the number of digits of the denominator is smaller than 3*n+4 (see [Bri 90]).

3. If the input maps result from an overlay themselves (thus containing rational coordinates), the same number of digits as in statement 2 is sufficient for the representation of the intersection points. This is due to the fact that no line segments connecting intersection points are introduced.

Under realistic assumptions rational numbers of finite precision are an easy, relative efficient and numerical exact coordinate representation for the map overlay. The presented algorithm and other plane-sweep algorithms are absolutely robust by this approach. For an efficient use of rational coordinates, an adequate support by hardware and system software is desirable.

3.4 Further applications of the algorithm

The presented algorithm can be used with little modifications for other problems. The *merge operation* is one of them, which is closely related to the map overlay [Fra 87]: Merge neighboring areas in one map representing the same thematic attribute. For example, such maps may result from a classification of the attributes or from an overlay with a non-injective overlay function. The neighboring areas with the same attribute can be merged by an algorithm similar to the presented overlay algorithm. The merge algorithm does not insert edges, which separate areas with the same attribute into the sweep line status. Thus, the resulting regions describe the merged areas.

The evaluation of the *intersection*, *union*, or *difference of polygons (with holes)* is a special case of the overlay problem. It can be done by using the following overlay functions $f_\cap$, $f_\cup$ and $f_\setminus$. To handle the polygons within the overlay function correctly we assign to each polygon the thematic attribute 1.

$$f_\cap(t_1, t_2) = \begin{cases} 1 & (t_1.A = 1) \text{ and } (t_2.A = 1) \\ r & \text{otherwise} \end{cases}$$

$$f_\cup(t_1, t_2) = 1 \qquad (+ \text{ merging of the results})$$

$$f_\setminus(t_1, t_2) = \begin{cases} 1 & (t_1.A = 1) \text{ and } (t_2.A = r_2.A) \\ r & \text{otherwise} \end{cases}$$

4 Analytical and experimental performance analysis

4.1 Analytical performance analysis

Let n be the total number of edges of all polygons and k be the total number of intersection points of all edges. In [MS 87] is shown that the problem of computing all intersections between two sets S and T of line segments in the plane, where no two segments in S (similarly, T) intersect, can be executed optimally in O(n*log(n) + k) time. Because overlay algorithms have to compute in any case the intersection points between the input maps this is also a lower bound for map overlay algorithm.

In the following, we consider the analytical performance of our overlay algorithm in detail. If no vertices coincide, the maximum number of event points is reached and is equal to n+k. Therefore the operation on the event point scheduler can be performed in O(log(n+k)) = O(log n) (because $k < n^2$). Because n is the total number of edges, the operation on the sweep line status can be executed in O(log(n)) time. The vertices of the polygon can be sorted in O(n*log(n)) time. If we assume that the number of edges attached to one event point is limited by a constant, the overlay can be performed in:

$$t\,(n,k) = O\,(\,(n+k) * \log\,(n))\ \text{time}$$

Thus, our overlay algorithm has the same time complexity as Nievergelt and Preparata’s algorithm for determining the regions of a self-intersecting polygon [NP 82].

4.2 Experimental performance analysis

We implemented our plane-sweep overlay algorithm in Modula-2. To examine the performance experimentally, we ran tests on a SUN workstation 3/60 under UNIX. Because of missing hardware and system software for rational coordinates, we used an 8 Byte floating-point representation, which was supported by the system. The implementation was developed to demonstrate the properties of the algorithm but it was not tuned for speed. Consequently, there is scope to speed up the overlay.

We performed the overlay between two input maps, where n is the total number edges of all polygons of both input maps. We ran four series of tests with maps consisting of (1) a regular net of areas to get a constant proportion p of k / n where k denotes the total number of intersection points, (2) areas covering the map completely which are generated by a tool, (3) tool-generated areas covering only 50 per cent of the map, and (4) real data. Test series 1 was performed with different proportions p. In test series, 4a two maps of administrative divisions of Italy were overlaid where one was translated by an offset. In test series 4b the state frontier of Uganda and lakes near by were overlaid. Typical input maps of the series are depicted in figure 9:

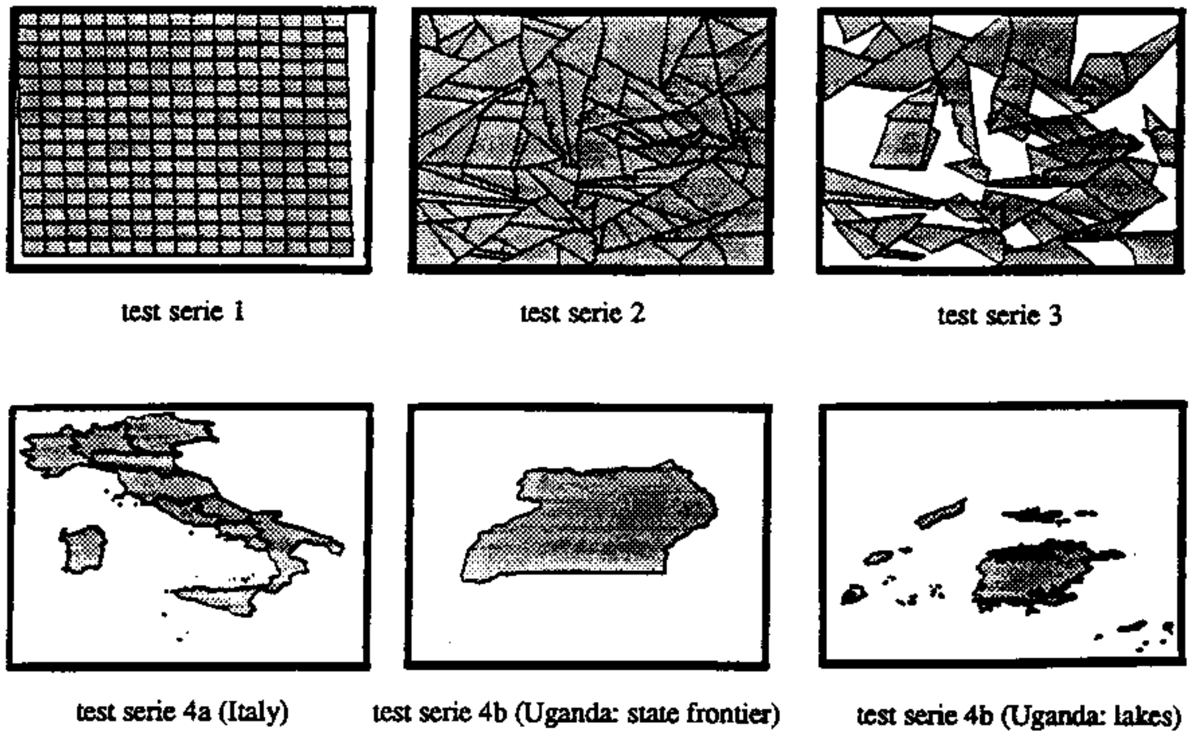
test serie 1
test serie 2
test serie 3
test serie 4a (Italy)
test serie 4b (Uganda: state frontier)
test serie 4b (Uganda: lakes)

Figure 9: Input maps of the test series

The results of the test series 1 are shown in table 1. t is the used CPU time in sec, which is needed to perform the overlay. Additionally, we want to determine the constant c of our map overlay algorithm hidden by the 0-notation (c = t / (n * ln n)).

Table 1: Results of the test series 1

serie 1a (p = 0.25):			serie 1b (p = 0.1):			serie 1c (p = 0.033):		
n	t [sec]	c [msec]	n	t [sec]	c [msec]	n	t [sec]	c [msec]
2048	25	1.61	2880	29	1.26	2160	18	1.11
4608	59	1.52	5120	53	1.22	4860	44	1.06
8192	112	1.51	8000	86	1.20	8640	83	1.06
10368	142	1.48	11520	130	1.21	11760	113	1.03
15488	227	1.52	15680	183	1.21	15360	147	0.99
21632	313	1.45	20480	246	1.21	19440	190	0.99
25088	363	1.43	25920	307	1.17	24000	246	1.02

The test series of table 1 demonstrate how the constant depends on the number of intersection points. An analysis of these tests results in the following function:

$t(n,k) = c' * (n+1.75*k) * \ln(n)$

The value of c' in the test series la to 1c is approximately 1.05 msec. We would like to emphasize that this constant is very small with respect to performance criteria.

In table 2 the test series 2 and 3 are depicted where c' is calculated additionally.

Table 2: Results of the test series 2 and 3

serie 2:

n	p	t [sec]	c [msec]	c'[msec]
13176	0.240	188	1.50	1.02
28837	0.154	372	1.26	0.97
30285	0.161	413	1.32	1.01

serie 3:

n	p	t [sec]	c[msec]	c'[msec]
6251	0.262	101	1.85	1.21
14179	0.184	221	1.63	1.20
15260	0.188	245	1.66	1.22

Series 2 and 3 demonstrate another dependency of the running time: If a large number of edges of the polygons coincide (see figure 9), the running time decreases. The reason is that the algorithm detects such edges and combines them.

Table 3 depicts the results of test series 4 with files of real data. In series 4a (Italy) varying administrative divisions and in series 4b (Uganda) varying resolutions are considered.

Table 3: Results of the test series 4

serie 4a (Italy):

division	n	p	t [sec]	c [msec]	c'[msec]
state	6666	0.012	67	1.14	1.12
groups of regions	9622	0.015	94	1.07	1.04
regions	11542	0.014	110	1.02	0.99
provinces	20378	0.023	194	0.96	0.92

serie 4b (Uganda, $p < 0.004$):

n	t [sec]	c[msec]
1852	16	1.16
8973	86	1.05
17829	180	1.03

The results of test series 4 demonstrate the validity of the previous results for real data.

5 The use of spatial access methods for map overlay

The database system of a GIS must support efficient query processing as well as efficient manipulation and combination of maps. To fulfill these requirements we assume that the database system uses suitable *spatial access methods (SAMs)* for the management of the database. In particular, this allows extracting the relevant parts from the seamless database (maps). In the following we assume that each map is supported by its own SAM because in GISs an efficient access to a map of one topic is desirable, e.g. land utilization or soil pollution.

An often used technique to store areas with SAMs is to approximate them by *minimal bounding boxes (MBBs)*. MBBs preserve the most essential geometric properties of geometric objects, i.e. the location of the object and the extension of the object in each axis. The query processing is carried out in two (or more) steps. MBBs are used as a first filter to reduce the set of candidates. The second step examines those candidate polygons by decomposing them into simple spatial objects such as convex polygons, triangles, or trapezoids [KHHSS 91b]. To test the polygons for intersection with a sweep line or query rectangle, MBBs are a sufficient first filter.

In the following, we assume that the SAM organizes the access to the objects of a database using a tree-like directory. Such access methods are adequate in handling non- uniform spatial data [KSSS 89]. The inner nodes are called *directory pages*; the leaves of the tree are *data pages*. The data and directory pages of a SAM define a partition of the data space. In our case a record in a data page consists at least (a) of a MBB, (b) of the value of the thematic attribute, and (c) of a polygon description or of a pointer to such a description depending on the size of the polygon, see [KHHSS 91a].

As mentioned in the introduction, the database and the maps in a GIS may be very large. Therefore, it is not useful to keep all overlay maps in main memory, especially not in multi user systems. In systems with a virtual storage manager the efficiency could decline by a large number of page faults.

Instead of processing the maps completely, it is more efficient to partition the maps and to carry out the overlay on these partitions. One approach is to partition the map using a uniform grid like in [Fra 89]. Obviously, this is not the best way because a non-uniform data distribution is not adequately handled by this approach. We will partition the map by using SAMs and the proposed overlay algorithm.

Another important reason to partition the maps is the running time of the plane-sweep overlay, which is more than linear. By partitioning, we reduce the number of polygons and edges for each overlay. Thus, it may be possible to carry out the total overlay faster.

5.1 Sweep-line partition

In the presented plane-sweep overlay, only those polygons are relevant which intersect the sweep line. Thus we have a criterion for partitioning by the algorithm itself: Only polygons intersecting the sweep line or close to the sweep line, are kept in main memory. In terms of SAMs, this means to read data pages from secondary storage as soon as the sweep line intersects them. We call this approach *sweep-line partition*.

5.1.1 Sweep-line partition and transformation

The sweep-line partition can be realized by the *transformation technique* [SK 88]. This technique transforms the coordinates of a 2-dimensional MBB into a 4-dimensional point. There are two representations of such points: (1) The *center representation* consists of the center of the rectangle (c_x,c_y) and the distance of the center to the sides of the rectangle (e_x,e_y). (2) The *corner representation* stores the lower left (x_1 ,y_1) and the upper right corner (x_2,y_2) of the box. The 4-dimensional points are stored by a suitable multidimensional point access method, e.g. the grid file [NHS 84], PLOP-hashing [KS 88], the BANG file [Fre 87], or the buddy tree [SK 90].

In the following, we use the transformation technique with corner representation. For the SAM of each input map an own sweep line is introduced. These sweep lines are oriented at the partition of the SAM. The pages to the left of or intersecting these sweep lines have already been read from secondary storage. Performing the map overlay, we must synchronize the overlay sweep line and sweep lines on SAM level for each input map. When the overlay sweep line overtakes one of the sweep lines of the SAMs, new data pages must be read from secondary storage by the corresponding SAM and the sweep line of the SAM is adjusted accordingly. An example is depicted in figure 10.

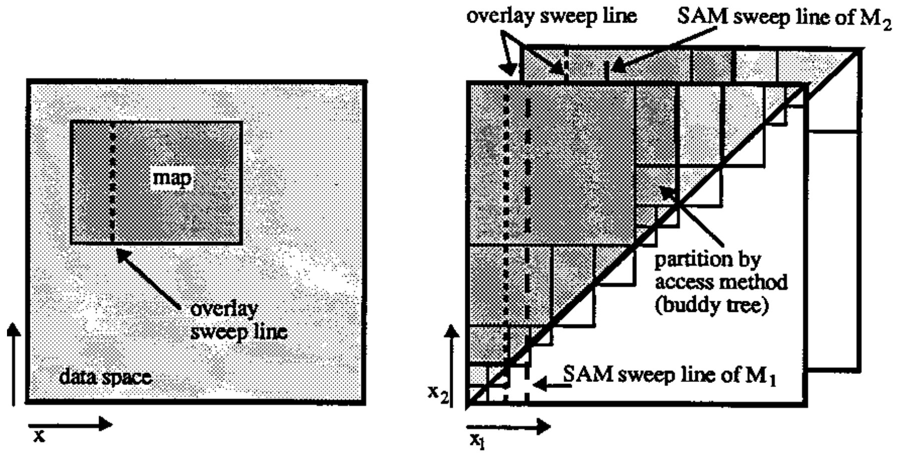

Figure 10: Sweep-line partition and realization by transformation (x-dimensions are shown only)

5.1.2 Performance

Using the sweep line partition reduces the number of page faults considerably because only those parts of the maps intersecting the sweep line reside in main memory. Minimizing the number of page faults during the overlay improves the overall performance.

This gain of efficiency is only slightly reduced by the following effect: Without partition, every required page is accessed exactly once. The pass through the tree of the SAM according to the sweep line may cause several accesses to the same directory page. However, the number of accessed directory pages is, compared to the total number of read pages, generally very small (considerable less than 1 per cent).

Compared to the complete overlay we observed that the CPU-time used for the partitioned overlay decreases about 10 per cent independent of the number of edges. The reason is the smaller average number of entries in the event point scheduler.

5.2 Strip Partition

Contrary to a partition oriented by the sweep line, an orthogonal partition is possible. This idea is used, for example in [Pul 90]. To support the plane sweep, it is sensible to divide the map into strips S_i, which extend over the whole map *(strip overlay)*. In the following, we assume proceeding from the top strip S_1 to the bottom strip S_M (see figure 11) The strip partition shortens the length of the sweep line, which decreases running time. The height of the strips may vary to adapt the partitions to non-uniform data distributions.

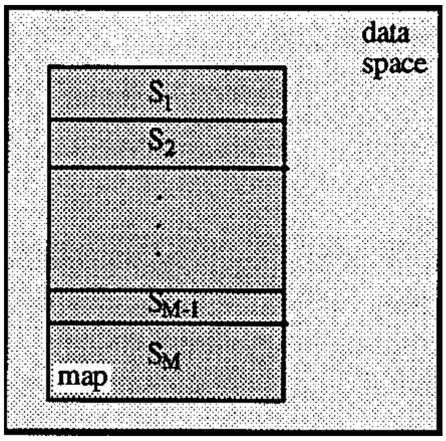

Figure 11: Strip partition

Some areas of a map may intersect more than one strip. To process these areas correctly, computed regions intersecting a strip, which is not processed, yet must be stored temporarily in a buffer. Such regions are an additional input to the next strip overlay. Thus, every area of an input map can be assigned to exactly one strip S_i and needs to be read from secondary storage only once.

As in section 5.1, we assume that all data pages are completely read. The access to the areas of one strip corresponds to standard region queries, which supply all polygons intersecting the strip. There is only one exception: Data pages accessed by previous strips are not read again. We call this kind of query *modified region query*. Such queries are performed very efficiently by the R*-tree [BKSS 90], a variant of the well-known R-tree [Gut 84]. This is caused by the minimization of area, margin, and overlap of directory rectangles in the R*-tree.

5.2.1 Generating an optimal strip partition

In the following, we want to describe how an optimal strip partition of the map is generated. An optimal strip partition adapts the strips to the distribution of the areas of the input maps, exploits the size of main memory, and avoids page faults. The strip partition is best supported by using an efficient SAM, such as the R*-tree.

As mentioned, the areas of each input map, which are simple polygons with holes, are approximated by minimal bounding boxes, which preserve the location and the extension of the areas. The number of bytes representing an area is assigned to the MBB. This is necessary because we cannot expect in GIS applications that each area is described by the same number of bytes. Each data page represents a set of areas. Thus, for each data page the number of bytes can be calculated which is necessary to store the data of this page in main memory. This information is stored in lowest level of the directory.

In a preprocessing step of the map overlay, we determine the data pages for all input maps, which intersect the map. Each data page of a SAM corresponds to a partition of the data space, e.g. the regions of the R*-tree are rectangles. These regions are sorted in descending order according to the highest y-coordinate of the region. Initially, the buffer is empty which stores areas, which are not performed completely by a strip, overlay. According to the order mentioned above the first k, regions are determined where the sum of bytes, which are represented by these k regions, is smaller than the size of main memory minus the size of the buffer. Thus, the first strip S_i is limited by the highest y-coordinate of the (k+1)st data page. The areas, which are not handled completely in the first strip overlay, will be stored in the buffer. The above procedure is iterated.

To illustrate this approach we present the following example where the size of main memory is restricted to 8 Megabytes (see figure 12).

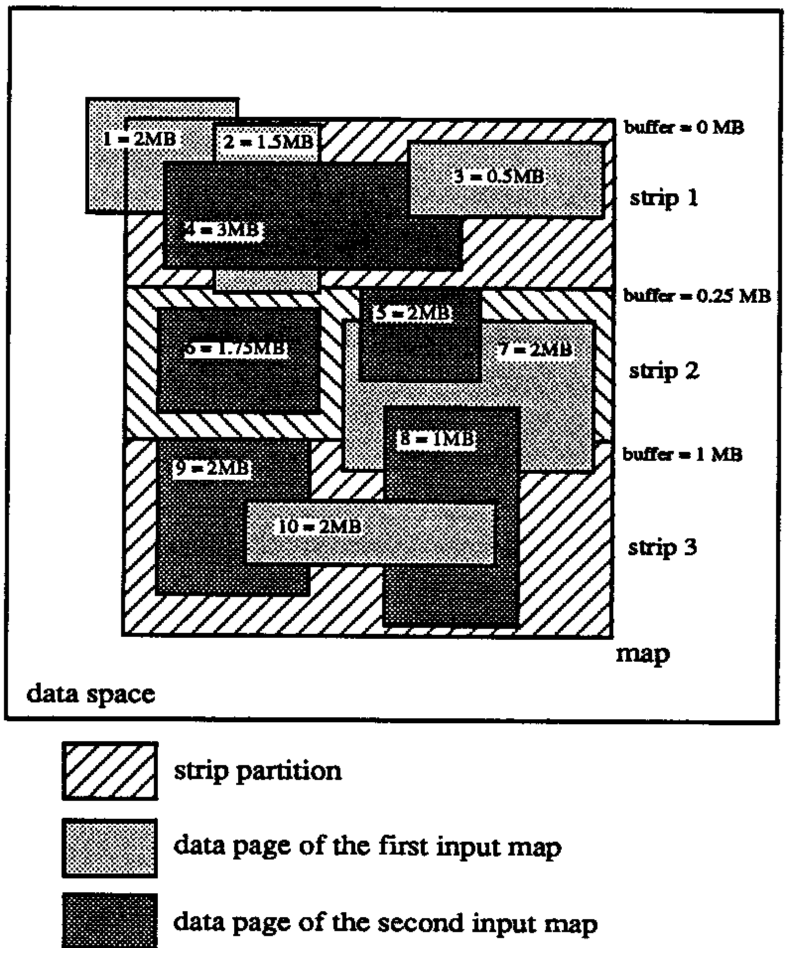

Figure 12: Example of generating an optimal strip partition

The numbers 1 to 10 of the data pages of the input maps indicate the order mentioned above. The size of the data pages 1 to 4 amounts to 7 MB. With the next data page the size of main memory would be exceeded (7MB+2 MB ≥ 8MB) and page faults could occur. Therefore, the first strip ends at the highest y-coordinate of the data page 5. Let us assume that after the first strip overlay 0.25 MB are stored in the buffer. Then the second strip can be extended until 7.75 MB are not exceeded. Thus, the data pages 5 to 8 are associated to the second strip. Finally, the data pages 9 and 10 and the regions stored in the buffer are accommodated in the third strip.

Generating the optimal strip partition is not time intensive because only directory pages are read to get the necessary information, such as the size of the data pages and bytes represented by the data pages. Only if the map overlay is performed data pages with the complete description of the areas are read from secondary storage. The ratio of read directory and data pages is smaller than 1% when performing the map overlay.

6 Conclusions

In this paper, we presented an efficient map overlay algorithm based on spatial access methods and computational geometry concepts, in particular on the plane-sweep paradigm. The marriage of these two areas was enabled by the property that the spatial access method supports the plane-sweep paradigm. Since plane-sweep processing generates results in sorted order, the spatial access method must be robust with respect to sorted insertions for storing these results. The operation map overlay was selected because it is the most important analysis operation in a GIS, e.g. Tomlin's map analysis package [Tom 90] is completely based on map overlay. Thus, the presented map overlay algorithm provides good analysis performance in GIS. Good analysis and retrieval performance are the most important factors for good user acceptance of a GIS. Thus, in our future work, we will design efficient algorithms based on spatial access methods for all retrieval operations. Performance improvements which exceed those realized in this paper by coupling spatial access methods and computational geometry, are feasible by using processors suitable for rational numbers and by implementing parallel GIS algorithms on parallel multi-processor systems. These issues are important goals in future work.

Acknowledgement

We thankfully acknowledge receiving real data representing national administrative divisions of the European countries by the Statistical Office of the European Communities. The data of test series 'Uganda' is taken from the World Data Bank II. Additionally, we would like to thank Holger Horn for making his map generator available to us.

References

[BKSS 90] Beckmann, N., Kriegel, H.-P., Schneider, R., Seeger, B.: The R -tree: An Efficient and Robust Access Method for Points and Rectangles. Proc. ACM SIGMOD Int. Conf. on Management of Data, 322-33 1, 1990

[BO 79] Bentley, J.L., Ottmann, T.A.: Algorithms for Reporting and Counting Geometric Intersections. IEEE Trans. on Computers, Vol. C-28, No. 9,643-647, 1979

[Bri 90] Brinkhoff, T.: Map Overlay of Thematic Maps Supported by Spatial Access Methods. Master Thesis (in German), University of Bremen, 1990

[Bur 86] Burrough, P.A.: Principles of Geographical Information Systems for Land Resources Assessment Oxford University Press, 1986

[Fra 83] Franklin, W.R.: A Simplified Map Overlay Algorithm. Proc. Harvard Computer Graphics Conf., 1983

[Fra 84] Franklin, W.R.: Cartographic Errors Symptomatic of Underlying Algebra Problems. Proc. Int. Symp. on Spatial Data Handling, Vol. I, 190-208, 1984

[Fra 87] A.U. Frank: Overlay Processing in Spatial Information Systems. Proc. 8th Int. Symp. on Computer-Assisted Cartography (Auto-Carto 8), 16-3 1, 1987

[Fra 89] Franklin, W.R. et at.: Uniform Grids: A Technique for Intersection Detection on Serial and Parallel Machines. Proc. 9th Int. Symp. on Computer-Assisted Cartography (Auto-Carte 9), 100-109, 1989

[Fre 87] Freeston, M.: The BANG file: a new kind of grid file. Proc. ACM SIGMOD Int. Conf. on Management of Data, 260-269, 1987

[FW 87] Franklin, W.R., Wu, P.Y.F.: A Polygon Overlay System in Prolog. Proc. 8th Int. Symp. on Computer-Assisted Cartography (Auto-Carte 8), 97-106, 1987

[Gün 89] Günther, O.: The Design of the Cell Tree: An Object-Oriented Index Structure for Geometric Databases. Proc. IEEE 5th Int. Conf. on Data Engineering, 598-605, 1989

[Gut 84] Guttman, A.: R-Trees: A Dynamic Index Structure for Spatial Searching. Proc. ACM SIGMOD Int. Conf. on Management of Data, 47-57, 1984

[KHHSS 91a] Kriegel, H.-P., Heep, P., Heep, S., Schiwietz, M., Schneider, R.: A Flexible and Extensible Index Manager for Spatial Database Systems. Proc. 2nd Int. Conf. on Database and Expert Systems Applications (DEXA '91), 1991

[KHHSS 91b] Kriegel, H.-P., Heep, P., Heep, S., Schiwietz, M., Schneider, R.: An Access Method Based Query Processor for Spatial Database Systems. Proc. Int. Workshop on DBMS's for geographical applications, Capri, May 16-17, 1991

[KS 88] Kriegel, H.-P., Seeger, B.: PLOP-Hashing: A Grid File without Directory. Proc. 4th Int. Conf. on Data Engineering, 369-376, 1988

[KSSS 89] Kriegel, H.P., Schiwietz, M., Schneider, R., Seeger, B.: Performance Comparison of Point and Spatial Access Methods. Proc. 1st Symp. on the Design of Large Spatial Databases, 1989. in: Lecture Notes in Computer Science 409, Springer, 89-114, 1990

[MS 87] Mairson, H.G., Stolfi, J.: Reporting and Counting Intersections Between Two Sets of Line Segments. Proc. NATO Adv. Study Inst. on Theoretical Foundations of Computer Graphics and CAD, 307-325, 1987

[NHS 84] Nievergelt, J., Hinterberger, H., Sevcik, K.C.: The Grid File: An Adaptable, Symmetric Multikey File Structure. ACM Trans. on Database Systems, Vol. 9, No. 1, 38-71, 1984

[NP 82] Nievergelt, J., Preparata, E.P.: Plane-Sweep Algorithms for Intersecting Geometric Figures. Comm. of the ACM, Vol. 25, No. 10, 739-747, 1982

[Oos 90] Oosterom, P.J.M.: Reactive Data Structures for Geographic Information Systems. PhD-thesis, Department of Computer Science at Leiden University, 1990

[PS 88] Preparata, E.P., Shamos, M.T.: Computational Geometry. Springer, 1988

[Pul 90] Pullar, D.: Comparative Study of Algorithms for Reporting Geometrical Intersections. Proc. 4th Int. Symp. on Spatial Data Handling, 1990

[SK 88] Seeger, B., Kriegel, H.-P.: Techniques for Design and Implementation of Efficient Spatial Access Methods. Proc. 14th Int. Conf. on Very Large Data Bases, 360-371, 1988

[SK 90] Seeger, B., Kriegel, H.-P.: The Buddy-Tree: An Efficient and Robust Access Method for Spatial Database Systems. Proc. 16th Int. Conf. on Very Large Data Bases, 590-601, 1990

[Til 80] Tilove, R.B.: Set Membership Classification: A Unified Approach To Geometric Intersection Problems. IEEE Trans. on Computers, Vol. C-29, No. 10, 874-883, 1980

[Tom 90] Tomlin, C.D.: Geographic Information Systems and Cartographic Modeling. Prentice-Hall, 1990

[Whi 78] White, D.: A Design for Polygon Overlay. Harvard Papers on Geographic Information Systems, Vol. 6, 1978

Appendix: Formal description of the plane-sweep overlay

A(e) (B(e)) are the A(B)-lists of e in the following. * denotes the concatenation of lists.

```
PROCEDURE PlaneSweepOverlay;

   VAR   x   : EventPointScheduler (Point,LeftEdges,RightEdges);
         y   : SweepLineStatus (Edge, A-list, B-list, RegionlD);
         P   : Actual event point;
         s   : Left edges of P; (sorted as in section 3.2 )
         t   : Right edges of P; (* sorted as in section 3.2 *)
         n,m : Number of entries of s and t respectively;
         h,l : neighbor edges of s[1] and s[n] in y;


   PROCEDURE TestForlntersection (edge1, edge2);

   BEGIN
      IF (edgel and edge2 intersect) AND (intersection point right of P) THEN
         Insert intersection point of edgel and edge2 into x;
      END;
   END TestForlntersection;
   PROCEDURE TestAndReportLists (A-list, B-list, P);

   BEGIN
      IF A-list points to the tail of B-list THEN
         Report polygon B-list * P;
      ELSE
         Concatenate: A-list * P * B-list;
      END;
   END TestAndReportLists;

BEGIN (*PlaneSweepOverlay*)

   (* Initialization *)
   x := Polygon vertices sorted by x-ordinates with attaching edges of the polygon;
   y := ∅;

   (* Pass *)
   WHILE x≠∅ DO
      (P,s,t) := Combined first event point of x;
      Delete (P,s,t) from x;
      n := number of edges of s;
      m := number of edges of t;
      (* Process left edges *)
      IF n=O THEN (* case l *)
         A(t[1]) = B(t[m]) := P;
      ELSIF n=1 THEN (* case 2 *)
         A(t[1]) := A(s[1]) * P;
         B(t[m]) := P * B(s[1]);
      ELSE (* case 3 *)
         IF m=O THEN (*case3a*)
            TestAndReportLists (A(s[1]),B(s[n]),P); (* hole in region *)
         ELSE (*case3b*)
            A(t[1]) = A(s[1])*P;
            B(t[m]) = P*B(s[n]);
         END;
         FOR i:=1 TO n-1 DO
            TestAndReportLists (A(s[i+1]),B(s[i]),P); (* region *)
            Delete s[i] from y;
         END;
         Delete s[n] from y;
      END;
      (* Process right edges *)
      IF m=O THEN (* case 1 *)
         h := former upper neighbor edge of s[1] from y;
         j := former lower neighbor edge of s[n] from y;
         Adjust IDs of regions between h and j in y;
         TestForlntersection (h,j);
      ELSIF m=1 THEN (* case 2 *)
         Insert t(1) into y;
         TestForlntersection (t[1], upper neighbor edge of t[1] from y);
         TestForlntersection (t[1], lower neighbor edge of t[1] from y);
      ELSE (* case 3 *)
         FOR i := 1 TO m DO
            Insert t[i] into y;
         END;
         FORi:-1 TO m-1 DO
            Set new region ID between t[i] and t[i+1] in y;
            B(t[i]) = A(t[i+1]) := P;
         END;
         TestForlntersection (t[1], upper neighbor edge of t(1) from y);
         TestForlntersection (t[m], lower neighbor edge of t[m] from y);
      END;
   END (*WHILE*)
END PlaneSweepOverlay;
```