NETHERLANDS GEODETIC COMMISSION
PUBLICATIONS ON GEODESY
NEW SERIES
ISSN 0165 1706
NUMBER 40

ADVANCED GEOGRAPHIC DATA MODELLING

SPATIAL DATA MODELLING AND QUERY LANGUAGES FOR 2D AND 3D APPLICATIONS

EDITED BY

MARTIEN MOLENAAR
SYLVIA DE HOOP

1994

NEDERLANDSE COMMISSIE VOOR GEODESIE, THIJSSEWEG 11, 2629 JA DELFT, THE NETHERLANDS
TEL. (31)-(0)15-782819, FAX (31)-(0)15-782745

CIP-DATA KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Advanced

Advanced geographic data modelling: spatial data modelling
and query languages for 2D and 3D applications
/ ed. by Martien Molenaar, Sylvia de Hoop.
Delft : Nederlandse Commissie voor Geodesie. - Ill. -
(Publications on geodesy. New series, ISSN 0165-1706 ; nr. 40)
With ref.
ISBN 90-6132-249-9
Subject headings: geographical information systems / geodesy.

PRINTED BY MEINEMA B.V., DELFT, THE NETHERLANDS

Contents

Foreword

The conference calendar of 1994 showed a coincidence of two international meetings with an overlap in their field of interest. These were the international symposium on Spatial Data Handling (SDH) in Edinburgh and the symposium of ISPRS Commission III in Munich, both in the week from 5 to 9 september. Because of the fact that omnipresence is still not one of the faculties of men, not even of GIS specialists, the Subcommission on GIS of the Netherlands Geodetic Commission took the initiative to organise a workshop where participants of both meetings could meet. This initiative was made in cooperation with members of the ISPRS Intercommission Working Group III/IV on "The Conceptual Aspect of GIS", but due to ISPRS regulations the meeting could not be put of the ISPRS calendar. A organising committee was formed in which the department for "Surveying and Remote Sensing" of the Wageningen Agricultural University took a major responsibility.

The organising committee decided to have a workshop with a limited number of participants (about 35) to have ample time for paper presentations and discussions. They selected the title "Advanced Geographic Data Modelling" for the workshop and the call for papers was for contributions on conceptual modelling for GIS, with topics of special interest like: data modelling, spatial query languages, dimensional aspects, multi scale problems and uncertainty and fuzzy data. Most of these are longer standing problems in GIS research, but they still require full attention of researchers because many questions are still open. That was also demonstrated by the fact that there was a large response to the call for papers. From the received extended abstracts the programme committee selected about 60% for presentation, the authors were invited to hand in full papers to be distributed in the workshop. There the papers have been presented and discussed in ten very lively sessions. Jan van Roessel from ESRI was invited for an opening address in which he treated the vector overlay problem in in relation to data modelling. He put this problem in the perspective of his 20 years of experience in data modelling in GIS. Donna Peuquet of the University of Pennsylvania reviewed the presented papers and the discussions in a wrap up at in the final session of the workshop. She discussed the presented research as a present state in the (short) history of the development of concepts in GIS. After the meeting the authors got the opportunity to update or rewrite their papers so that they could take advantage of the discussions in the workshop to improve their texts. These final papers you find collected in these proceedings.

The fact that the workshop got the short working title AGDM'94 means that we hope that more workshops in this set up will be organised in the future. Most of the participants asked for a continuation, so we have to see how that can be realised. We certainly hope that somebody is willing to take the initiative.

The Subcommission on GIS of the Netherlands Geodetic Commission would like to thank the organising committee and the programme committee of the workshop for the work done to make AGDM'94 a success. We specially like to thank Sylvia de Hoop who played a major role in the organisation. We also would like to thank the sponsors who were very generous in providing funds to cover a substantial part of the costs of the workshop. But most of all we like to thank participants of the workshop for their active contributions in both presentations and discussions. Through their contributions we had a very lively meeting of a very good quality.

Martien Molenaar
Chairman of the Subcommission on GIS
Netherlands Geodetic Commission

Organization and Sponsoring

The AGDM '94 Workshop is organized under the auspices of:

Netherlands Geodetic Commission
Wageningen Agricultural University (WAU), Department of Surveying and Remote Sensing, The Netherlands
TNO Physics and Electronics Laboratory, The Netherlands
University of Twente, Department of Computer Science, The Netherlands
Netherlands Centre of Expertise for Geographical Information Processing (NexpRI)
National Center for Geographic Information and Analysis (NCGIA), USA

ORGANIZING COMMITTEE

Martien Molenaar	Wageningen Agricultural University, The Netherlands
Sylvia de Hoop	Wageningen Agricultural University, The Netherlands
Peter van Oosterom	TNO Physics and Electronics Laboratory, The Netherlands
Annita Wilschut	University of Twente, The Netherlands

PROGRAMME COMMITTEE

Peter Burrough	University of Utrecht, The Netherlands
Eliseo Clementini	University of l'Aquila, Italy
Paolino Di Felice	University of l'Aquila, Italy
Max Egenhofer	NCGIA, University of Maine, Orono, USA
Gregor Engels	University of Leiden, The Netherlands
Peter Fisher	University of Leicester, UK
Sylvia de Hoop	Wageningen Agricultural University, The Netherlands, co-chair
David Mark	NCGIA, University of New York, Buffalo, USA
Martien Molenaar	Wageningen Agricultural University, The Netherlands, chair
Peter van Oosterom	TNO Physics and Electronics Laboratory, The Netherlands
Terence Smith	NCGIA, University of California, Santa Barbara, USA
Annita Wilschut	University of Twente, The Netherlands

SUPPORTING AND SPONSORING ORGANIZATIONS

Geodelta, The Netherlands
Geops, The Netherlands
Logisterion, The Netherlands
Netherlands Federation for Earth Observation and Geo-Information
Netherlands Geodetic Commission
Siemens Nixdorf Informationsystems, The Netherlands
Smallworld Systems, The Netherlands
Synoptics, Integrated Remote Sensing & GIS Applications, The Netherlands
Unisys, UK
Wageningen Agricultural University, The Netherlands

Opening Address

THE VECTOR OVERLAY PUZZLE: WHERE DO THE PIECES FIT?

JAN W. VAN ROESSEL

Environmental Systems Research Institute, 380 New York Street,
Redlands Ca 92373

ABSTRACT. This paper reviews the authors experience with vector overlay and structuring. He first becomes acquainted with vector-structuring and overlay processing in the development of a forest management GIS, begun almost 20 years ago. After the demise of this system he develops an experimental overlay processing system using plane sweep and adaptive grid algorithms, while refining the attribute propagation ideas developed in the earlier system.

This work leads to his current involvement with the ARC/INFO system, where the principles of attribute propagation are applied to buffer zone generation and in structuring overlapping areal units for a newly developed region feature class.

Currently, the author attempts to integrate these and other concepts into a point-attribute model in which some of the puzzle pieces begin to fall into place.

1. AN EARLY SYSTEM

In 1970 I joined the Berkeley, California, West-Coast office of the Earth Satellite Corporation, a newly established firm, offering consulting services in remote sensing. After some years, I became involved in the development of a "forest management information system," to be marketed to the forest industry. The design for this system, later known as LANDPAK, was begun in 1976, while its development stretched into the early eighties.

1.1 Features

This early system was transaction oriented featuring a B+tree storage system featuring shadow data blocks with rollback and commit. The spatial objects were points, lines and polygons organized in layers. Polygon boundaries were digitized once, but stored twice. Layers were organized in arbitrary boundary tiles termed "control units." The system had a data definition facility and a query language for polygon overlay. Each spatial attribute record had an associated attribute record that could handle the basic primitive data types as well as other data types such as vectors and matrices. Polygon overlay would result in secondary layers that would be recursively related to the primary data layers.

1.2 Query Language

The query language for polygon overlay handled "within layer" selection expressions nested in "between layer" boolean overlay expressions. An example query (van Roessel, Langley and Smith, 1978) is the following:

```
1. SELECT LAYER = COMMERCIAL THINNING $
2. IN AREA = BEAR CREEK $
3. WITH OPTIONS = MINIMAL SEARCH, DELETE INTERIORS $
4. FOR WHICH $
5. IN LAYER = COVERTYPE LAYER $
6. FOREST TYPE EQ 'C' AND STAND TYPE EQ 'EA' AND (SPECIES(-9) EQ 'PP' OR
7. SPECIES(-9) EQ 'DF' ) AND (ORIGIN DATE GE 1/1/1985 AND ORIGIN DATE LE 1/1/1940)
8. AND DENS INDEX GE 80 $
9. AND $
10. IN LAYER = SLOPE CLASS LAYER $
11. CLASS NUMBER EQ 1 $
12 END $
```

where lines 5, 6, 7, 8 form a covertype selection statement and lines 10 and 11 a slope class layer selection statement, while line 9 calls for a spatial intersection between the two selected sets. Between layer statements could be parenthesized the same as within layer statements. One might say that the above query is in "layer normal form." The within layer statements would be compiled and be passed on to the selection sub-system, while the between layer statements would drive the overlay system.

1.3 Strip System

The overlay methodology invented for the system consisted of dividing the spatial data for the layers in a tile into strips, where each strip is free of intersections. Completely horizontal line segments were eliminated by raising one end by a minuscule amount. Line segments in a strip were represented by integer references to the parent segments, and the total set of strips was called a "tableau." The approach thus had certain analogies to a grid system.

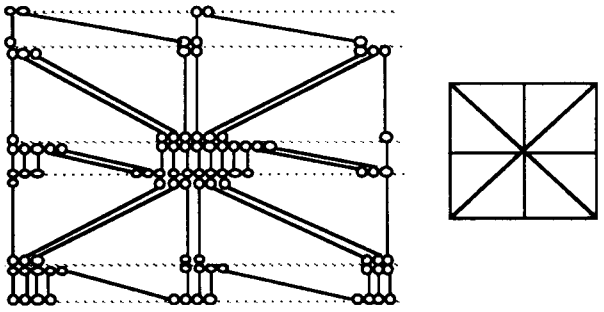


Figure 1. Square with diagonals and corresponding tableau.

Figure 1 shows a square with diagonals and the corresponding tableau. The vertical strip widths are out of proportion to better show its structure. Horizontal lines are raised at the left with an added vertical line segment. As can be seen, this leads to some traffic jams, especially in the middle of the figure. The structuring method handled these configurations by organizing adjacent line segments into line bundles. Polygon extraction would follow the border segments in the line bundles. The tableau was built by a process of "strip splitting" using a tolerance system that would guarantee a minimum vertical strip width, thus limiting the number of strips that could be generated.

1.4 Attribute Propagation

One step in the overlay was the classification of the tableau. As the end result of this process a use-counter for each strip line segment would be set for the polygon extraction process, indicating whether the segment was to be used once, twice or not at all. The use-counters were computed by propagation attributes along a strip. Each line segment is assigned an attribute called a "glump key." The word glump was used by the Codasyl Development Committee (1974) to indicate a partitioning of a set by a function defined on the members of a set. The glumping function defines the glump key. Here, the glump key is a layer number indicating the polygon layer of which the line segment is a part.

The propagation consisted of three steps: (1) propagating glump keys along the line segments, making a glump key "stack" for each segment, (2) evaluating a boolean condition (overlay instructions generated from the query) against each line segment stack, and (3) computing the use counters from the stack. This process is demonstrated in Figure 2.

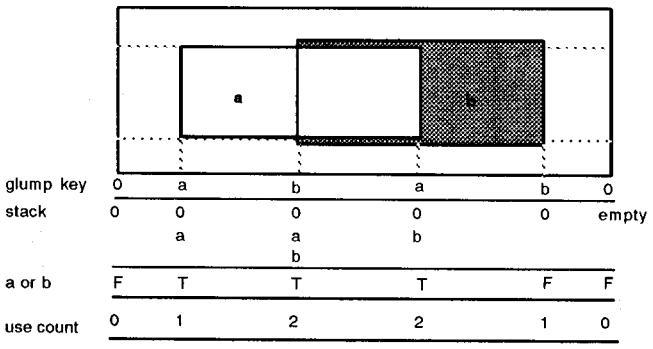


Figure 2. Line segment classification of vertical segments for the union of two overlapping rectangles

The use counters are computed by setting F= 0 and T= 1, and adding the truth values for a line segment and the line segment to its left.

The union example in figure 2 would result in three polygons with the middle two vertical segments used twice for a double boundary representation. Another type of union however, would return only one rectangle by setting the use count k to mod(k, 2). Thus, line segments interior to the union would be deleted. A boolean condition can expressed in reversed Polish notation and can be evaluated by a finite state machine, where each instruction results in a current state. This lead to the the idea of specifying interior line segments to be deleted for each state.

The system employed two polygon extraction methods at various times. The first used a line segment tracking system, while the second one operated on a sweep basis. The sweep system used the idea of constructing "lobes" from the strip trapezoids and then assembling these into polygons.

Another important task was the propagation of "record keys," similar to the propagation of "glump keys." Record keys would uniquely identify parent polygons so that for any derived polygon a parent history would be available, to be stored in a secondary attribute record.

2. AN EXPERIMENTAL SYSTEM

The EarthSat West-coast office was closed in 1981, and I went to work at the Applications Division of the USGS EROS DATA Center, in Sioux Falls, South Dakota, a facility known for LANDSAT data management and raster processing. To follow up on earlier overlay work, I began to develop an experimental overlay system to further develop previous ideas and experiment with new techniques in line with other published research (van Roessel, 1991).

The objectives for the development of this system were to: (1) develop dynamic plane-sweep methods, (2) use a better system of tolerances, (3) introduce point clustering, (4) improve the previous method of line segment classification, (5) improve the previous structuring method and (6) provide output in alternative forms, for instance with chain-node topology.

2.1 Plane-sweep using binary trees

Rather than create a static tableau, the experimental system uses two dynamic binary trees, one for vertices in the Y-direction, and one for line segments in the X-direction. In the vertical direction vertices are sorted such that in a sequential scan they are grouped as "half nodes" and the corresponding line segments are correctly ordered within these half nodes. In the X direction the line segments are uniquely ordered by a relation "to the right of" (van Roessel, 1991). Line segments are inserted into the X-tree and deleted from the X-tree, driven by a scan of the vertices in the Y-tree. In a bottom to top sweep, a line segment is inserted when its bottom vertex is encountered and deleted when its top vertex appears.

2.2 Data Normalization and Clustering

The thesis by V.J. Milenkovic (1988) entitled "Verifiable Implementations of Geometric algorithms Using Finite Precision Arithmetic" is a milestone in dealing with round-off errors in finite precision arithmetic as applied to overlay processing. Milenkovic develops the process of data normalization as a basis for robust geometric algorithms. As summarized in Milenkovic, 1989, the first three rules of data normalization are:

1. No two vertices are closer than ε.
2. No vertex is closer than ε to an edge of which it is not an endpoint.
3. No two edges intersect except at their endpoints.

The tolerance ε is based on an analysis of the computations involved in the determination of point-line distance and depends on the number of operations and the round-off error (see also Pullar, 1994)

In the experimental system, coordinates are integers, while intersection computations are performed in floating point arithmetic. In that case, the round-off error resulting from the conversion of floating point coordinates to integers is much larger than the errors accumulated in the floating point operations.

The processes of "vertex shifting" and "edge cracking" guarantee that the rules of data normalization are observed. Edge cracking consists of subdividing line segments when rule 2 is violated, and when line segments intersect. Vertex shifting occurs to make rule 1 the invariant. It leads to the expanded concepts of vertex clustering where vertices are grouped together using other clustering criteria as well. In the experimental system line segment cracking is followed by vertex clustering. Clustering may use different weights to move certain vertices or to anchor others. A second line segment cracking pass may sometimes have to follow a clustering pass, when the clustering method introduces additional intersections. In summary, the total situation is more complex than portrayed by Milenkovic. One must consider three types of tolerances: (a) floating point computation errors, (b) floating point to discrete grid rounding errors, and (c) clustering tolerances. These may all be of different magnitudes.

2.3 Adaptive grids

Line segment cracking is carried out through the use of an adaptive grid, a data structure advocated by Franklin (Franklin et al, 1989). Line segments are coarsely gridded in a grid that is adapted to the size of bounding rectangle of the data. The line segment is assigned to the cells that it passes through. Conflicts with other line segments in the same cell are resolved through line segment cracking when necessary. The cell size must be coordinated with the clustering tolerance to catch all cracking situations. Line segments are entered into the adaptive grid as driven by a Y-tree scan. Whenever a line segment is cracked, the line segment is removed from the grid, the line segments resulting from the cracking are inserted into the Y-tree, and the current insertion point is moved back to the lowest Y-coordinate of the cracked segment, where everything is still "safe." The process then resumes, so that the action moves back and forth over the adaptive grid until the results are stable.

2.4 Overlay Steps

Line segment cracking, vertex shifting, Y- and X- binary vertex and line segment trees, vertex clustering, are some of the puzzle pieces that belong to the overlay process so far. The following questions arise: (1) is there a tolerance system in which one does not have to back up to the low Y- coordinate of the cracked segment, (2) is there a tolerance system in which the one cracking pass is sufficient and (3) is there a clustering method using weights that is compatible with the first two objectives. In the experimental system a very simple but very fast transitive closure clustering method is used. This method has the undesirable characteristic that elongated clusters are replaced with a single point.

2.5 Structuring Method

A new structuring method was developed for the experimental system based on the previously mentioned half nodes and lobes. An intermediate data structure is developed from input polylines, from which the traditional topological components, such as polygons, rings, chains and line segments can be retrieved through pointer lists. The intermediate structure is developed in a plane-sweep process, in which line segments are connected into "half-nodes," and half-nodes are connected into monotone polygonal subdivisions termed "lobes." Lobes can be defined such that connections (splits and joins) are forks, so that "connection" can be easily treated as an equivalence relation. This allows connecting lobes to be assigned to equivalence classes, which define polygons. The equivalence classes may be used to remove topological

singularities before final structuring, but singularities may also be consistently structured. The fact that lobes always have a left- and right hand side leads to a very efficient algorithm for collecting the outer and inner rings for polygons.

2.6 Attribute Propagation

Attribute handling in the experimental system is based on the methods developed in the LANDPAK system, but instead of having line segments with a single glump key we assume that each line segment has a left and right glump vector, where each element in the vector is of type boolean. Instead of maintaining a single "stack" in the propagation process, we collect left and right "bag" vectors.

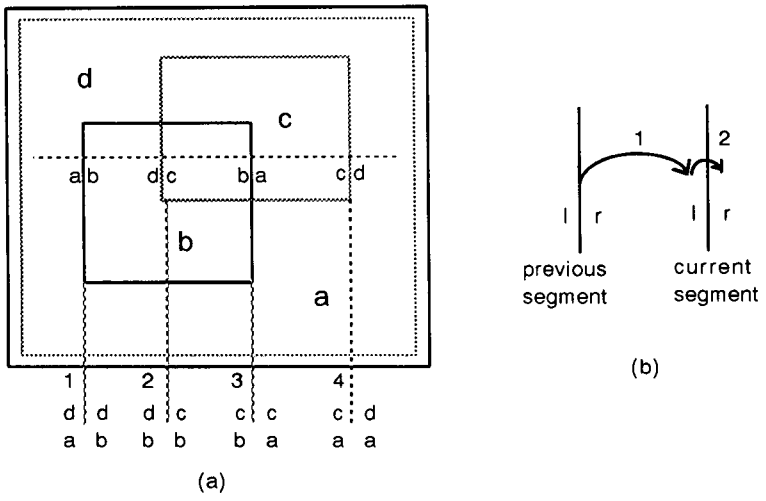


Figure 6. Systematic attribute propagation along a transect

The advantage of having left and right glump vectors, is that line segments initially may have different left and right attributes.

The attribute propagation is performed in two steps, as shown in figure 6, beginning at the right side of a segment: (1) transition from the right side to the left side of the next segment and (2) transition from the left side to the right side of the same segment. For the first step, the bag may be transferred unchanged. The rule for the second step is to remove any attribute from the bag that is identical to a left attribute of the segment under consideration, and to add the attribute on the right. Denoting the two segments involved in the propagation cycle as p (previous) and c (current), and left or right elements of the bag and glump vectors as $b_l(x)$, $b_r(x)$, $g_l(x)$, and $g_r(x)$, respectively, the two step process can be expressed using the boolean operators:

Step 1: $b_l(c) = b_r(p)$ or $g_l(c)$

Step 2: $b_r(c) = (b_r(p)$ and not $g_l(c))$ or $g_r(c)$

A user devised function with a boolean result type operates first on the left and then on the right attribute bags and returns either a left and then a right *use classification* that is either true or false, reflecting whether the attributes satisfy a required condition. Next, the left and right use

classification combination is used to decide whether the line segment is exterior, on the boundary, or interior to the desired condition, as shown in the following truth table. Exterior line segments are deactivated. The deactivation of interior line segments is optional.

Left Use	*Right Use*	*Status*	*Disposition*	*Use Count*
T	T	Interior	Optional	2
T	F	Right	Keep	1
F	T	Left	Keep	1
F	F	Exterior	Dispose	0

The attribute propagation and line segment classification method described so far can be used to perform arbitrary combinations of boolean operators operating on any number of coverages or subsets of coverages, depending on how a coverage is defined. But vector processing tasks in geographic information systems (GIS) are not confined to pure boolean combinations, but include other functions that have accumulated a specialized terminology in the development of GIS systems, such as merge and dissolve, drop line, erase, clip, update, and paste up. Some of these terms were introduced at an early state in GIS development, and survive to date (Tomlinson, 1972). Many of these specialized tasks can be performed with the proposed methodology, if the following extension is made to the line segment classification using the Polish notation.

Line segments can be classified into interior, exterior, and left and right boundary classes. The extension of the method relates to the handling of interior line segments. When the union operation is performed for example, both figures 7e and 7f satisfy the query (a *and* b) *or* c, but in figure 7f the interior boundaries have been dissolved. For the Polish method, at every step, as shown in figures 7a-e, there is a selected area that corresponds to the state at the top of the stack.

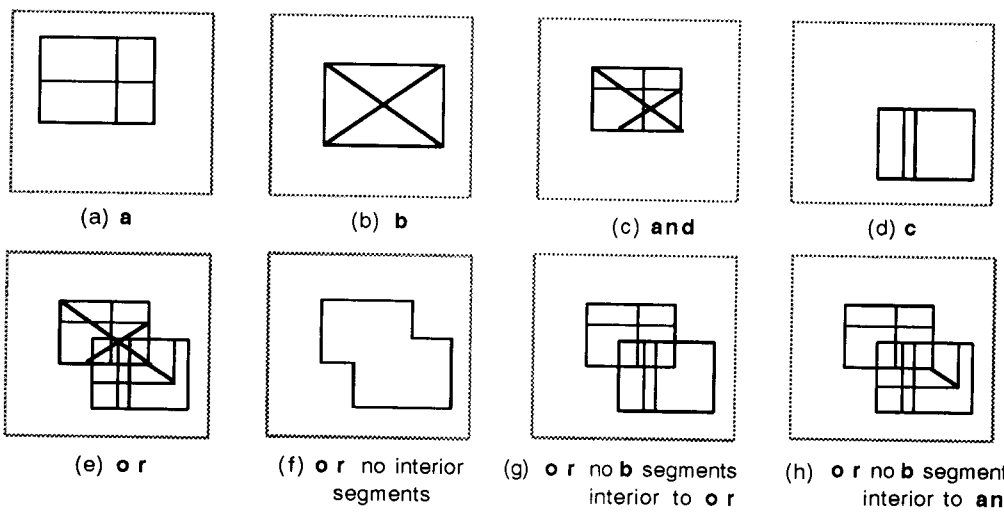


Figure 7. Areas corresponding to top stack states for execution of (a and b) or c.

The or of figure 7e is not merely the union of figures 7c and 7d; instead, the line segments of b are visible in 7e. The reason is that the overlay process examines all coverages *simultaneously* as opposed to the usual two at a time approach.

Each stack state also corresponds to a node in an expression tree. The merge and dissolve shown in figure 7f shows how interior line segments can optionally be removed. The crux of the method for the extension to other vector tasks is that this decision can be made at every stack state or node of the execution tree. Moreover, the possibility exists to be selective by selecting only those interior segments that correspond to one of the original input glumps. For example, figure 7g shows the final result with the line segments of b removed interior to the *or*, and figure 7h shows the final result with removal of b line segments interior to the *and*.

2.7 The Control Code

This approach was implemented for the developed overlay processor that is driven by Polish code, where each instruction has the following syntax, as expressed in BNF (Backus-Naur Form):

<control instruction> ::= <instruction> <glump key> '/'
<interior delete list>
<instruction> ::= SEL | AND | OR | NOT
<glump key> ::= integer
<interior delete list> ::= {glump key}

where the "{}" brackets indicate zero, one or more repetitions, and the "|" vertical bar indicates a choice. An additional requirement is that *<glump key>* for the operators *AND*, *OR* and *NOT* must be 0. The *SEL* operator tests the presence or absence of the specified glump key in the left or right line segment bag and accordingly puts either true or false at the top of the stack.

The following table lists the control instructions for a number of common and not so common GIS vector operations involving two maps

Operation	*Code*	*Comment*
Clip	SEL 1; SEL 2; AND 0/2;	The outline of 2 clips 1
Erase	SEL 1; SEL 2 / 1 2 ; OR 0;	The outline of 2 is superimposed on 1
Identity	SEL 1; SEL 2; AND 0; SEL 1; OR 0;	2 is merged into 1, with outline of 1
Intersect	SEL 1; SEL 2; AND 0;	Traditional intersect
Union 1	SEL 1; SEL 2; OR 0;	Keep interior boundary segments
Union 2	SEL 1; SEL 2; OR / 1 2	Delete interior boundary segments (merge and dissolve)
Update 1	SEL 1; SEL 2 / 1; OR 0;	Update 1 with 2
Update 2	SEL 1 / 2; SEL 2; OR 0;	Update 2 with 1
Not	SEL 1; SEL 2; NOT 0; AND 0;	Keep area of 1 that is not in 2
Erase 2	SEL 1 / 1; SEL 2 / 2; OR 0;	Erase boundaries interior to 1 and 2 but not interior to both.

2.8 Line on Polygon Overlay

The title of this paper reflects the fact that overlay in GIS systems is more than polygon on polygon overlay. Frequent references to polygon overlay in the literature ignores the fact that line on polygon overlay is also an important topic. The method described above can be extended to also cover line on polygon overlay. In addition to being able to specify which *interior* line segments should be deleted at a particular stage of the evaluation, one can then also specify the *exterior* line segments that are deleted. The previously described grammar can be extended to have both exterior and interior deletion lists at each stage of the boolean evaluation. This allows simultaneous mixed line-polygon multi-layer overlay. Interior lines will become singularities when exterior surrounding lines are deleted, and thus one must address the question of how to structure singularities. This is another very important puzzle piece.

3. A WIDELY USED SYSTEM

I joined ESRI's software development team in 1991, and became responsible for maintenance and change of the ARC/INFO topological structuring and overlay related software. Part of this work was the development of a new feature class for overlapping and potentially non-contiguous areal units (regions).

3.1 ARC/INFO Overlay

The ARC/INFO overlay related programs date back to the early eighties. The reader is referred to Guevara (1983) for an overview of some of the ARC/INFO overlay procedures. The programs are based on techniques earlier developed for the Odyssey system (Guevara, 1985).

The topological structuring methodology used in ARC/INFO has similarities to the techniques described for the experimental system. After line segment intersection, topological structuring is performed in three stages: (1) a sort of the intersected line segments, (2) processing the sorted line segments by nodes generating arcs (chains), and (3) polygon generation by connecting the arcs for a polygon. Step 2 is a plane-sweep step, processing vertices by descending y-coordinate and within equal y by ascending x- coordinate. Arcs are generated by processing nodes. Nodes are full nodes and may connect "old" and "new" line segments. Old line segments have a vertex that is not at the current node, already processed as a previous node.

3.2 Buffer generation

One of the problems that I encountered at ESRI was that of making the existing approach for generating buffers around lines more efficient. The existing approach at that time (still in use for generating buffers around polygons) was to generate "raw buffers" (closed shapes, but with cusps and other irregularities), to intersect these raw buffers, convert to polygons, generate a label in every polygon, check the label distances of the original features, and finally dissolve all interior boundaries.

The problem was solved by developing a method in which it is known in a very early stage which line segments are interior to the final buffer. Thus, the interior line segments can be discarded in the polygon building phase. Rather then build a larger number of sliver polygons, the final buffer polygon is built directly.

Knowing the line segments that are interior to a buffer is an attribute propagation problem. The line segments of the original raw closed buffer shapes are assigned left and right keys for the interior and exterior of the shape, for instance 0 / 1 (1 meaning inside). Attribute propagation is then a type of arithmetic that keeps track of the number of overlapping shapes for each line segment. When a number of raw buffers overlap, the part that is common to all receives a key equal to the number of overlapping shapes. By incrementing and decrementing these keys final buffer boundaries will receive a key of 0 / n where $n >= 1$. Unwanted line segments will have keys n / m, where $n, m > 0$. The propagation process within a node is shown in Figure 8.

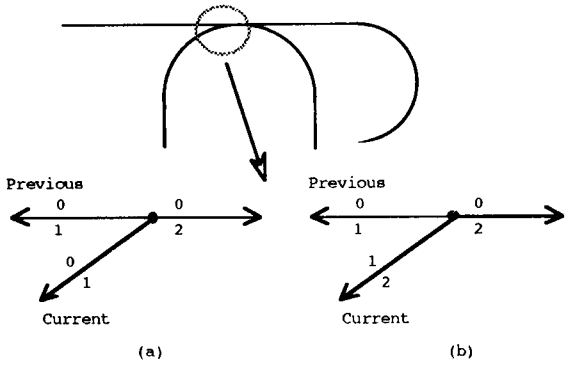


Figure 8. Key propagation within a buffer node: (a) before propagation, (b) after propagation.

Figure 8a shows the situation before propagation, while figure 8b shows the result after propagation. The transfer is from the left side of *Previous*, to the left, and then to the right side of *Current*. The transfer expressions are:

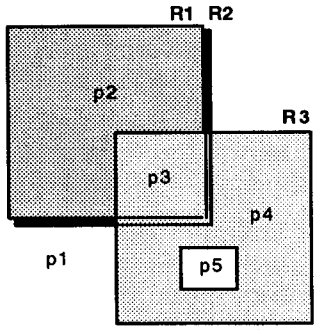
Differential = Current/Left - Current/Right
Current/Right = Previous/Left
Current/Left = Current/Right + Differential.

3.3 Overlapping areal units

Yet another implementation of attribute propagation, using a similar mechanism as employed for buffer zones found its way in the method for structuring a new ARC/INFO composite feature class called regions (van Roessel and Pullar, 1993).

In the past many ARC/INFO users have implemented some type of scheme to deal with overlapping data by implementing systems using the Arc Macro Language (Gross, 1991). The most sophisticated system of this type has been a cadastral parcel mapping system developed by ESRI-UK. Invariably, all these schemes have used the device of a cross-reference file. This file stores the relation between the overlapping units and the base polygons. Figure 9 shows the use of a cross reference file

It was decided that it was necessary to start providing basic support for overlapping data. We realized that overlapping areal units can also be defined in terms of arcs, and designed a composite feature called a region based on both arcs and polygons. The term region seems appropriate because a region can also have multiple disjoint components.



Region	Polygon
1	2 3
2	2 3
3	3 4

Figure 9. Cross-reference file example

One can think of a region's explicit relationship with arcs as the "geometry" of regions, while the relationship with polygons constitutes the "topology" representing the vertical relationship, as explicitly represented in the RXP file. In the life cycle of regions in ARC/INFO one can think of the arc and polygon as a region's "legs." At times one of the legs is missing and must be recreated from the other. Building the RXP file from arcs organized into rings, while at the same time building the underlying polygons, is part of the topological structuring process for regions.

There may be many region types in a single coverage. Each type has its own attribute schema and cross reference file. The types are called "subclasses." One may think of the region subclasses as layers that are integrated into a single coverage.

3.4 Building the cross reference file

The attribute propagation process used to create the RXP file is more similar to the process used in the LANDPAK system then the one of the experimental system. We start from an input state in which regions are assigned to arc line segments. Each line segment may be a border segment of multiple regions and therefore may have a set of associated regions. To make matters more complicated, regions may belong to different subclasses, each subclass with its own schema.

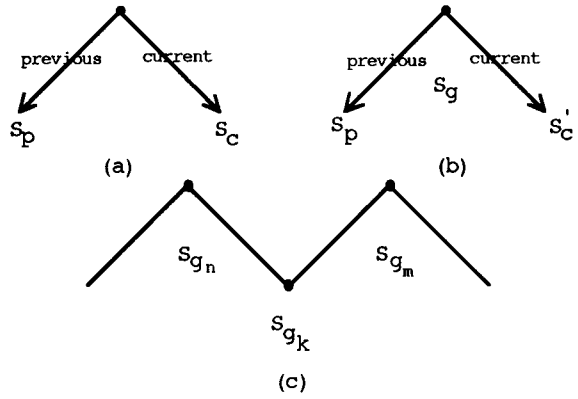


Figure 10. Set propagation for regions: (a) before propagation, (b) after propagation, (c) propagation for confluence of two gaps.

Region subsets are propagated within nodes as with the buffer scheme.

Again we sweep around the node, and progress from the *previous* to the *current* line segment. If we denote the region set of the previous line segment by S_p and the set of the current segment by S_c then the objective is to obtain the region set for the gap between the two line segments, S_g, and the modified set for the current line segment S'_c. The transfer expressions are the following

$$S_g = S_p$$
$$S'_c = S_c \oplus S_g$$

where $\oplus$ is the symmetric difference. Thus regions in the gap set are "canceled out" by similar regions in the current line segment set, and the current line segment is changed accordingly.

When two gaps come together, as show in Figure 10c, we have that

$$S_{g_k} = S_{g_m} \cup S_{g_n}$$

The process continues, using an equivalence class scheme to finally arrive at the complete set of regions for each polygon under construction. This relationship is then inverted to derive the RXP file.

3.5 REGIONQUERY

In addition to developing the methodology to structure overlapping data, we also needed to answer the question of how to handle overlapping data in overlay type queries. The overlap factor introduced a new puzzle piece in an already complicated puzzle. The solution that was developed consists of two strategies.

The basic building block in ARC/INFO is the coverage. A coverage is either of the point, line or polygon type, while a polygon coverage contains lines in the form of arcs and points in the form of nodes. Regions exist in polygon coverages as region "sub-classes." Each subclass has its own attribute schema. If the regions in a subclass do not overlap, then a region coverage with multiple subclasses is a set of integrated "layers" with the base polygons as the largest common units.

The classical overlay operations in ARC/INFO are UNION, INTERSECT and IDENTITY. In addition there are "data management type" overlays, such as clip, erase and dissolve.

The first strategy developed was that of region subclass inheritance for general two-layer overlay. Let $Cov(A)$ denote a coverage with a set of region subclasses A. F_b is a binary operator that produces one output coverage from two input coverages. The region functionality for this type of function is:

$$Cov3(C) = Cov1(A)\ F_b\ Cov2(B)$$

where $C = A \cup B$. If A and B contain an identical subclass s, each with region sets P_S and Q_S, then C will have subclass s with a region set $R_S = P_S \cup Q_S$. With identical subclass, we mean that the subclass bears the same name and has the same attributes, and therefore has the same schema. The function will fail for subclasses with identical names and different attributes. This strategy converts the traditional overlay function into region subclass integrator functions. For example, we may begin with two polygonal coverages, and convert the polygons into regions

using a POLYREGION function followed by a union on the two coverages. Two region subclasses identical to the two polygon coverages will then present in the output coverage.

The second strategy was to develop a REGIONQUERY function that exploits the vertical topology in the form of the RXP file for a subclass.

REGIONQUERY performs boolean selection against any number of region subclasses. In addition to selection, it also produces output regions that are homogeneous with respect to a number of user specified output items.

REGIONQUERY usage can be represented as:

SELECT<out_subclass> {WITH <in_subclass.attribute...in_subclass.attribute>}

WHERE <logical_expression>

where *<out_subclass>* is the output subclass, pre-existing or to be created, *<in_subclass.attribute>* is an attribute of *<in_subclass>* that will be added to the schema of *<out_subclass>*. The set of these attributes forms the output attribute list. The list is optional.

The *<logical_expression>* is a completely general boolean expression where the operands are either constants or attributes of the form *<in_subclass.attribute>*. A special pseudo item *$subclass*, meaning "where the subclass exists" (no-zero record number) can also be used. The logical expression used defines the "territory" qualifying for the query, while the output attribute list defines the "granularity" of the output. Territory in the region's case also extends to the vertical dimension, and the same is true for granularity, because identical overlapping regions with identical output attribute values are collapsed to a single region.

The REGIONQUERY function also has an option to produce contiguous output regions.

The model under which REGIONQUERY operates is the following. Each polygon in the input coverages has a number of regions for a given subclass of which it is a part. These are the regions "above" the polygon. The polygon and the regions above it form a subclass "stack."

If only a single subclass is involved in the query, the selection expressions are evaluated for each member of the stack, and if true, the polygon is selected to be a part of a candidate output region corresponding to that member of the stack for which the expression is true.

If multiple subclasses are present, and a single polygon has multiple subclass stacks with more than one region, REGIONQUERY will combine the stacks in the form of a Cartesian product, and evaluate the attributes according to the logical expressions for each member of the product.

Output items are assigned to each selected Cartesian product combination and a dissolve is performed to make the attribute value combinations unique for each output region.

4. AN INTEGRATED POINT-ATTRIBUTE MODEL

The above two strategy approach was the resulting of adapting to existing software as well as developing new software. It presents yet additional puzzle pieces to be fitted into an overall model. Without the constraint of previous functionality one might ask: is there a general model that covers the above approach that provides a more general insight into the overall process, including the use of overlapping units.

In the LANDPAK query method described early on in this paper, the query was decomposed into within layer and between later selection, where the between layer selection consists of overlay processing. Later on, we realized that an additional post-overlay selection step is required, for instance to deal with comparisons such as < or > between attributes in different layers. Thus, the overlay query process can be represented by a general three step process. The ARC/INFO strategies for regions reflect the last two steps: overlay processing using any of the classical overlay commands, and post-overlay processing in the form of REGIONQUERY. In van Roessel (1994) we try to develop a more general model, as follows.

The general selection expression can be expressed in domain or tuple relation calculus, with the map-layers as the relations. It contains an expression on the attributes of the map-layers using boolean and comparison operators. This selection expression is translated into three separate expressions that come to play in three different stages of the overlay process:

1. Pre-selection expressions that are applied to each of the map-layers before the actual overlay.
2. An overlay expression governing the spatial combining of the map-layers.
3. A post overlay selection expression, operating on the integrated result.

In the following we shall assume that the first step has taken place, and shall only be concerned with steps 2 and 3. For these steps, we present a set of general expressions for overlay that are applicable to four types of areal units (combinations of overlapping and non-overlapping, and contiguous and non-contiguous). These expressions encompass various types of traditional overlay operations, such as the intersection, identity and union operations. They may also be used to convert between different types of areal units. Here, we restrict ourselves to two layer overlay, but in van Roessel (1994) the approach is generalized to *n* layer overlay. The model is predicated on the fact that the areal units in a set of registered maps can be decomposed into smaller units that are common to both sets of maps. On a conceptual level these can be points, or they can be largest common units, such as the base-polygons for regions in ARC/INFO. Operations can then be expressed in terms of relations.

The two layer overlay process begins with two map-layers X' and Y' of type $\{[\chi,S]\}$ and $\{[\chi,T]\}$. Here S and T are conventional attribute schemas while, χ represents any of the four areal unit types. The [ ] brackets indicate a tuple type, and the { } brackets indicate a set type. Steps 2 and 3 above are expanded into steps 1-6.

Step 1. We begin by unfolding the two input maps:

$$X = \varphi^{-1}(X'), \quad Y = \varphi^{-1}(Y')$$

The unfold operator is the inverse of the fold operator defined in van Roessel (1994). The term folding is borrowed from Lorentzos and Johnson (1987), who introduced the concept for a temporal relational algebra (van Roessel, 1993). The fold operator is expressed as :

$$(Y, R_y) = \varphi_\tau(X)$$

where X is a base relation consisting of either unique points of possible duplicated (overlapping points) and is of meta-type $\{[\omega,S]\}$ where ω represents the point type.

The fold operator φ_τ can produce either contiguous or non-contiguous area units from points. Y has meta-type $\{[\chi,S]\}$ and R_y is a cross-reference relation of meta-type $\{[\chi,\omega]\}$. The meaning of the unfold operator φ^{-1} is expressed in relational algebra as:

$$X = \pi_{\omega,S}(Y \textit{ njoin } R_y)$$

Here *njoin* is the natural join (projected equi-join, where the output attributes are the set union of the input attributes). The interpretation for the point-set model is that the areal unit-attribute relation is joined with an areal unit-point relation to obtain a point-attribute relation. In the region polygon case, one may substitute the base polygon for the point. In that case we get a polygon-region attribute relation, as in the case of REGIONQUERY. This is the equivalent of the sub-class region stack mentioned earlier.

Step 2.

$$X_c = \pi_{\omega,S\cap T}(X), \quad Y_c = \pi_{\omega,S\cap T}(Y)$$

In the second step we find the attributes, including spatial attributes, that are common to both input layers and project both unfolded relations on these attributes. In most cases this will only be spatial attributes, but there is no reason to treat other attribute different from spatial attributes. The only difference between the tuples of the projected relations at this point is then the input map that they belong to. For the LANDPAK and experimental system type overlay, this step reflects the line segment glump key assignment, where a glump key represents the input layer.

Step 3.

$$Z = (((\sigma(X_c,Y_c)) + \text{join } X) + \text{join } Y)$$

The schema compatible sets of step 2 are combined using a set-combinatorial expression $\sigma(X_c,Y_c)$ This expressions is equivalent to the LANDPAK query between layer expression. For that example we have that $\sigma(X_c,Y_c) = X_c \cap Y_c$. For ARC/INFO region processing there is no equivalent because the query expression is not parsed into within and between subclass expressions. All base polygons are evaluated. The model points out however, that is not the most efficient method. If a between subclass expression could be developed from the REGIONQUERY *WHERE* clause, then X_c and Y_c can be considered the sets of the base polygons that represent the subclass "footprints." These are combined according to $\sigma(X_c,Y_c)$ to yield a restricted set of base polygons that is to be evaluated further.

The resulting set of points (or base polygons) is then joined back to the unfolded input maps X and Y. For points that are duplicated in X or Y because of overlap we then get the Cartesian product of these records in the output relation Z. In REGIONQUERY this is accomplished by forming the Cartesian product between attribute records of different subclasses over a base polygon. In the remainder of the expression for step 3, +join means right natural outer join.

Step 4.

$$Z' = \sigma_{exp}(Z)$$

The next step is to apply the post-overlay selection expression. In the case of REGIONQUERY, this means selecting Cartesian product attribute record combinations over a

polygon. In the LANDPAK system it meant having to do a secondary query on the result of a previous overlay. An example is the following query

```
1. SELECT LAYER = HIGH EROSION HAZARD $
2. IN AREA = BEAR CREEK $
3 WITH OPTIONS = MINIMAL SEARCH, DELETE INTERIORS $
4. FOR WHICH $
5. IN LAYER = SOILS SLOPE OF 10/6/77, 2 $
6. SOILS *EHR(SLOPE*CLASS NUMBER) EQ 'H' OR
7. SOILS *EHR(SLOPE*CLASS NUMBER) EQ 'E'
8. END $
```

where soils and slope are intersected in the transaction of 10/6/77, and this transaction then references the computed erosion hazard by indexing an erosion hazard rating vector (EHR) by the slope class number.

Step 5.

$$W = (\varphi_{\tau}\pi_{H}(Z')) \text{ njoin } \pi_{SUT}(Z')$$

The last and final step involves the fold operator . This operator takes the point relation and "folds" it into a relation consisting of the final type of desired areal units. The folding is done with respect to a set of homogeneous attributes. Interior boundaries are resolved within unique attribute value combinations of these attributes. The resulting areal units may be contiguous or non-contiguous depending on the fold operator type used.

In terms of the LANDPAK system, the clause 'DELETE INTERIORS' meant removing the interior boundaries relative to a homogeneously attributed area.

In van Roessel (1993) it was suggested that a FOLD clause be added to a SPATIAL SQL to make queries such as:

```
1. CREATE COVER goodsites AS
2. FOLD soils*, landuse*
3  SELECT soils*, landuse *
4. FROM soils, landuse
5. WHERE soils.loc = landuse.loc
6. AND soils.suitability = 3
7. AND landuse.lutype = 300;
```

Here the select set of attributes happens to be the same as the homogeneous attributes used for folding. This is always true in the ARC/INFO REGIONQUERY function, but need not be so, as shown by the expression given at the beginning for this step, where the fold result is joined with all the attributes of both input maps. This is the most general solution.

4.1 Query Optimization

A general point-attribute model for spatial query execution involving vector overlay seems useful, and helps in fitting some of the puzzle pieces. The process has many similarities to relational query modification and decomposition (Mayer, 1988). We did not address the question of how to convert a general selection expression into "layer normal form."

The above model also barely begins to address optimization questions. What algorithms can be used to for instance to arrive at optimal set operation expressions (Sheng and Sheng,1990).

However, in a practical context, it remains to be seen whether such optimizations would be cost effective, given the frequency and the nature of the queries posed. Brute force solutions on fast machines may be adequate.

5. CONCLUSION

In this paper we traced our personal involvement with vector overlay. The ideas, techniques, and unsolved problems are part of a larger puzzle that has only been partly assembled. Even if one could complete the mental puzzle then another step is to put the integrated components into widespread use. A number of problems and solutions were discussed in this paper, but some problems could not be mentioned here. For instance polygon on line overlay where the lines form a non-planar graph and may cross without forming nodes.

The future of polygon overlay is also uncertain. Rather than use plane-sweep methods one can think of a methodology in which for each unit inserted, the topology is updated on the fly, such as may be used for interactive editing. Recently Guting has proposed a scheme of realms (Guting, 1993) where points and lines are maintained in a fully intersected realm. The advantage of such a realm is that after each modification certain topological invariants hold, so that a very consistent processing system can be developed that is computationally robust.

With computational power ever increasing, raster processing is becoming far more important for modeling and may eventually replace vector use. On the other hand, for certain applications such as cadaster and ownership, vector use will still be preferred. Another question is whether fuzzy boundary processing is better performed with raster or vector data, and how this can be integrated into vector data handling.

REFERENCES

Codasyl Development Committee, 1974. An Information Algebra: System Analysis Techniques, Couger, J. D., and Knapp, R. W., eds, Wiley and Sons, New York.

Franklin, W.R., C. Narayanaswami, M. Kanhanhalli, D. Sun, M. Zhou and P.Y.F. Wu, 1989. Uniform Grids: A Technique for Intersection Detection on Serial and Parallel Machines," In: Proceedings, Auto-Carto 9, American Society for Photogrammetry and Remote Sensing, pp. 100-109.

Gross T., 1991. Modeling Polygons Located at Several Positions on a Z-Axis as a Single Planar Map. In: Proceedings, Eleventh Annual ESRI User Conference, Vol. 2, pp. 5-20.

Gueting, R.H., 1993. Realm-Based Spatial Data Types: The ROSE Algebra. Informatik Berichte Nr. 141, FernUniversitaet, Postfach 940, D-5800 Hagen, Germany.

Guevara, J.A., 1983. A Framework for the Analysis of Geographic Information System Procedures: The Polygon Overlay Problem, Computational Com-lexity and Polyline Intersection. Ph.D. dissertation, Geographic Information Systems Laboratory, State University of New York at Buffalo.

Guevara, J.A., 1985. Intersection Problems in Polygon Overlay. Auto-Carto 7, Digital Representation of Spatial Knowledge, Washington D.C., March 11-14.

Lorentzos, N.K. and R.G. Johnson, 1987. A Model for a Temporal Relational Algebra, In: Proceedings of Conference on Temporal Aspects in Information Systems, Sophia-Antipolis, France, North Holland Publishing Company.

Milenkovic, V.J., 1988. Verifiable Implementations Of Geometric Algorithms Using Finite Precision Arithmetic. Ph.D. Thesis, Carnegie Mellon University, Pennsylvania.

Milenkovic, V.J., 1989. Verifiable Implementations of Geometric Algorithms Using Finite Precision Arithmetic. In: Geometric Reasoning, edited by Deepak Kapur and Joseph L. Mundy, The MIT Press, Cambridge, Massachusetts.

Pullar, D., 1994. A Tractable Approach to Map Overlay. Unpublished Ph.D. Thesis, University of Maine, Orono, Maine.

Sheng, J.M. and O.R. Liu Sheng, 1990. Optimizing Geometric Set Operations Queries. In: Proceedings of the Fourth Symposium on Spatial Data Handling, 1990, Zurich, Switzerland.

Tomlinson, R. F., ed., 1972. Geographical Data Handling, Symposium Edition, International Geographical Union Commission on Geographic Data Sensing and Processing for the UNESCO/IGU, Second Symposium on Geographic Information Systems, Ottawa, Vol. 2, pp. 825-829.

van Roessel, J.W., P.G. Langley and T.D. Smith, 1978. Timber-Pak - A Second Generation Forest Management System. In: Integrated Inventories of Renewable Natural Resources: Proceedings of the Workshop, Jan 8-12, Tucson, Arizona.

van Roessel, J.W., 1991. A New Approach to Plane-sweep Overlay: Topological Structuring and Line Segment Classification. Cartography and Geographic Information Systems, Vol. 18, No. 1, pp 49-67.

van Roessel, J.W., 1993. Conceptual Folding and Unfolding of Spatial Data for Spatial Queries. In: Towards SQL Database Language Extensions for Geographic Information Systems. Ed. V.B. Robinson, H. Tom. National Institute of Standards and Technology Report NISTIR 5258, Gaithersburg Maryland. pp. 133-148.

van Roessel, J. W. and D. Pullar, 1993. Geographic Regions: A New Composite GIS Feature Type. In: Proceedings AutoCarto 11, Minneapolis, Minnesota. American Society for Photogrammetry and Remote Sensing and American Congress on Surveying and Mapping, Bethesda, Maryland. pp. 145-156.

van Roessel, J.W., 1994. An Integrated Point-Attribute Model for Four Types of Areal GIS Features. In: Proceedings of the Fifth Symposium on Spatial Data Handling, 1994, Edinburgh, Scotland.

Semantic Data Models

FORMAL SEMANTICS FOR THE INTEGRATION OF IMAGE DATA INTO GEOGRAPHIC INFORMATION SYSTEMS

MARK GAHEGAN
Dept. of GIS, Curtin University, PO Box U 1987
Perth 6001, Western Australia
tel.: +619 351 3309, fax: +619 351 2819
e-mail: mark@cs.curtin.edu.au

ABSTRACT

This paper describes a formal means of specifying the relationship between image data, and the spatial descriptions of geographic objects (*features*) derived from such image data, and used ultimately within a GIS. As such it forms part of an ongoing research project to create a combined GIS and Remote Sensing (RS) environment capable of supporting multiple spatial descriptions of a feature and of extracting these spatial descriptions from images in an automated *goal-directed* manner. This is achieved by examining the current task and its context and choosing (or creating) a description of the targeted features that exactly suits this task. A formalism based on the IFO semantic data model and Object Retrieval Calculus is presented to describe the semantics of feature extraction from a multi-source, multi-temporal image database.

1.0 INTRODUCTION

There are many difficult problems to solve before GIS can take full advantage of the vast amounts of image data available of the earth's surface. One example, in the area of image processing and computer vision, concerns the recognition of meaningful features[1] from the raw image data (Wang, 1988). However, once these features have been recognised it is vital that the GIS knows exactly what is meant, or implied by each one. There is a need to better understand the deep issues; that is, the semantics of the problem must be analysed and understood before the interpretation of imagery can have any real meaning.

A problem with the application of computer vision is the implication that a feature can be precisely and unambiguously defined by some process, and this definition then placed in a GIS. Unfortunately such an approach is a gross simplification of the real situation. That which defines a feature as distinct from its surroundings is often unclear, and in some cases, a single interpretation of a scene upon which people would agree is often

[1]Throughout this paper real-world geographic objects of interest are referred to as *features*. Informally, a feature is a simple geographic entity such as a town, a lake or a road.

not realisable[2]. It would seem that the task for which a feature is to be used has a great influence on how that feature might best be represented spatially.

The ability to utilise imagery directly in GIS in a more flexible manner can lead to several advantages, such as the availability of regular and frequent updates (weather permitting) and a reduced reliance on expert intervention. However, the incorporation of imagery leads immediately to several other problems in addition to that of actual image interpretation:

- A feature of interest may appear in more than one image and hence have more than one spatial representation. This points to the need for a mechanism to select representations appropriate in various situations.
- Images are captured at various times. A temporal dimension is added to the problem. Features may change, indeed the analysis of this change may be of prime importance.
- Images may be captured by different remote sensing platforms. Accordingly, a variety of expert knowledge and extraction methods are needed, along with details of their applicability in different circumstances.
- Many interpretations of the same image may be possible. Different image interpretation and feature extraction methods applied to the same image will produce different results.

To summarise, each feature can legitimately be described by several different geometrys, covering particular capture dates, sensors, interpretation procedures and uses. Each valid description is termed a *spatial representation*. The semantic issues involved in the construction and selection of appropriate spatial representations for a feature form the subject of this paper.

1.1 Spatial Representations and Context

As a general point, any extracted spatial representation is likely to have a limited applicability, governed by the factors mentioned above. These factors, when taken together, form a *context* within which a particular representation is valid. Outside of this context, the representation is likely to have limited or perhaps even no use. For example, a representation of a feature drawn from 1991 satellite data may be quite unsuitable for describing the current situation, and indeed may be entirely redundant if more recent imagery is available from the same source. Understanding the context over which a spatial representation is appropriate is a central theme to this work. A full description of such context is developed through Sections 2 and 3 below.

1.2 Differences between image data and GIS data

Much of the spatial data used in GIS have been formed by the process of digitisation. Such data is already in *feature* form, that is, individual objects of interest have been described or extracted by some manual or semi-automated method. In other words, a process of scene understanding (segmentation and classification) has already been carried out. In contrast, image data originates from a remote capture or scanning device, in the form of a raster, often with several layers corresponding to reflectance in different electro-magnetic spectra. Within an image there are no identified features, but rather a

[2]For example, if there are n geologists, there are at least $n+1$ opinions as to how the data should be interpretted.

snapshot of a scene encoded by a set of multi-valued pixels. To extract features from such an image requires further processing, with the use of scene understanding techniques (Tailor et al., 1986).

It can be seen that these two types of data are not at the same conceptual level and are consequently problematic to combine. It is important that the user of a GIS operates at a consistent level with respect to data (Gahegan, 1994). If image data is directly combined with feature data (e.g. within an overlay operation) it is unclear what the type of the resulting spatial description(s) is. Is it an image, or a feature, or a set of features? Hence there is a need for transforming the image data into feature form before it is used in a GIS.

1.3 The Integration of GIS and Remote sensing

The integration of GIS with remote-sensing systems (RSS) has been the subject of much recent research (Davis & Simonett, 1991), (Dobson, 1993), (Ehlers et al., 1991). However the work to date concentrates on the issue of system architecture, and not on the semantic issues of how the data is transformed and what the transformations really mean in an information sense.

An examination of established commercial GIS and RSS reveals that integration is at present primitive. These two technologies, although developed from a common thread, have diverged to such an extent that integration is an afterthought not a pre-requisite of the system architecture. However, some recent progress towards integration has been made. For example ER-Mapper[3] can display ARC-INFO coverages geo-referenced to image data.

The interpretation of image data is usually a precursory activity before it is entered into a GIS. Such an approach has two major weaknesses discussed below:

1. The communication lines between the two systems are poor. Specifically, GIS tend to ignore all output from image processing systems except the resulting classified overlay from which geographic features are drawn.

As an example, consider the tasks involved in producing a classification of a remotely-sensed scene for use in a GIS. The raw image data entered into an RS package are typically re-projected, sometimes re-scaled and then registered to some co-ordinate frame. Next a classifier is applied. Results may be expressed probabilistically (that is each pixel has a posterior probability of belonging to a particular class). To make the output suitable for GIS, some kind of *featurising* process occurs. Regions are formed by clustering together neighbouring pixels with high probability for the same class, sometimes followed by a smoothing operation to remove localised variations. Through all of the above steps it is possible to keep a track of the meta-data such as the date of image capture, the sensing platform and the uncertainty of the result (Burrough, 1987). When the derived feature boundary data is imported into a GIS, all of this meta-data is literally thrown away. The GIS data model has no means whatsoever of dealing with it.

[3]A popular remote-sensing package available from Earth Resource Mapping, Perth, WA.

2. The feature-boundaries obtained from image analysis are highly sensitive to the methods employed in their formation; the classifications being either quite specific in their applicability, or else too generalised. The details of the methods used are not available to the GIS (or its user) in any formal sense, neither is there any given indication of what particular uses the classification was designed for. The GIS user has no choice but to either accept the data at face value, or to request that a remote-sensing expert produces a more suitable interpretation. As a consequence, there is an alarming tendency amongst GIS users to ignore all issues relating to the accuracy and applicability of data and a belief that, if a polygon closes, then it has no associated errors.

The advantages to be gained from integration are therefore:
1. The ability to include the meta-data associated with the process of image analysis within the GIS data model. This is turn allows for a more comprehensive and qualified result when subsequent operations are performed on the data. Uncertainty estimates can thus be associated with the each stage of the entire data handling process from capture to output. A study of uncertainty is beyond the scope of this paper, but is being investigated by other researchers (Hunter & Goodchild, 1993), (Fisher, 1994). The architecture described in the latter sections of this paper has been deliberately designed to handle uncertainty, and each stage in the featurisation process has uncertainty properties.

2. The behaviour of the image analysis and feature extraction methods can be controlled and directed by the GIS. The GIS therefore has the power to govern exactly how feature boundaries are to be derived, and hence can supply the most suitable boundaries for the task at hand. That is, a task-oriented means of feature extraction is *driven* from the current query or operation. With the addition of expert knowledge the process of feature extraction can be automated to some extent.

1.5 System Architecture

In order to achieve full integration of image data within GIS, the following functionality must be developed:

(i) Image interpretation methods
Functions applied to image data are grouped together into methods, where a method is designed to accentuate or detect the presence of a particular feature-type in the data. The result of applying an interpretation method is an image view (that is a view of the image made for a purpose). Methods are often a combination of both low-level image processing algorithms and high level reasoning. Before a method is used it must first be configured with a set of parameters relating to a particular feature-type and sensor. These parameters include a variety of expert knowledge concerning the bands to use and typicality measures such as training set values. Further details are beyond the scope of this paper, but are described in Gahegan & Flack, (1994).

(ii) Control strategy
A means of choosing or creating spatial representations appropriate to the current task. The control strategy is discussed in detail in Section 2.

(iii) Feature extraction methods
Extraction methods attempt to isolate particular features from image views. The resulting spatial representation must then be somehow associated with the feature it describes, at execution time. Feature extraction is discussed in Sections 3.4 and 3.5, and makes use parameters in a similar manner to image interpretation.

The above list represents additional functionality not currently found in either GIS or RSS. The aim here is to allow the remote sensing component to be entirely controllable from within the GIS. In addition, the GIS must take from the RSS both the derived spatial representations and their associated meta-data.

Many different architectures have been proposed for combined GIS/RS systems, a good summary is given by Glenn et al. (1992). There are two possible architectures that could support the ideas described in this paper, namely fully integrated and closely coupled. A fully integrated solution involves the functionality of the two systems being merged into a single suite of software, designed as a single entity. The resulting system is integrated at the physical, conceptual and external level[4]. A closely coupled solution keeps the two systems independent, but they are interfaced at the conceptual, as opposed to the external level. In either case the tasks involved with image interpretation are under direct control of the GIS. The approach taken here is to design and develop a single environment from scratch. As such the work forms a natural extension to research carried out on image database systems (Grosky & Mehrotra, 1992).

1.6 Formalisms adopted

An object based approach is very natural for this type of application, since it attempts to model the changing behaviour of real world objects and their associated spatial and aspatial properties. Object-oriented conventions are used in this paper. Unfamiliar readers are directed to Worboys et al, (1990) where an overview of object-oriented concepts, as they apply to GIS, is given.

At the conceptual layer the GIS described here operates on geographic objects called features. Each feature is defined independently from any appearance in spatial data, in other words it does not depend for its existence on any overlay or image[5]. Features are grouped by their type, with each type described by a class within the conceptual data model. The class is a collection of methods, data structures and parameters to govern the behaviour of a particular type of object. Class descriptions are augmented here by knowledge regarding the recognition of features in remotely-sensed scenes, and by methods to facilitate their extraction.

A formalism based on the IFO semantic data model (Abiteboul & Hull, 1987) is presented to describe the structure of the system. The IFO model has been used by many researches as a means of expressing object-oriented concepts within data models (Roberts et. al., 1991), (Worboys et al., 1990). Object Retrieval Calculus (ORC) (Worboys et al., 1990), is then used to detail the semantics of a coupling between image

[4]For a full discussion on the three layer ANSI/SPARC architecture mentioned here (as used in relational databases and GIS design) see for example Korth & Silberschatz (1991).
[5]In many GIS currently available, the converse is true; the geometry *defines* the feature. It is to the geometric description that an identifier is added to act as an index into a table of aspatial attributes.

data and feature descriptions. ORC is an extension of Domain Relational Calculus (DRC) (Lacroix M and Pirotte A, 1977); an overview of the relationship between DRC and the more traditional relational algebra is given by Ullman, (1988).

ORC (and DRC) work in the following manner. The object-types to be retrieved (O) form the target list and are associated with free variables (x) that are said to range over the domain of each object type.

$$\{x_1: O_1, x_2: O_2, \ldots, x_n: O_n\}$$

Each free variable is then instantiated with corresponding objects from the corresponding class. A qualifying clause is used to place restrictions on the retrieved objects, and is defined by relationships between the various object properties and other functions, constants or objects. Further free variables may be introduced in the qualifying clause to support these relationships. When dealing with composite objects (that is, formed by aggregation or association), it becomes necessary to use their properties (which of course are also objects) in order to test certain conditions. This can be achieved by the use of functions that test for a certain condition and return either TRUE or FALSE. Examples of such queries are given in Worboys et al. (1990).

To provide ORC with a means of examining the properties of an object explicitly, two additional operators have been added into ORC. Specifically, these operators reflect the two ways in which complex objects are assembled, by aggregation or association. They allow free variables ranging over objects to be constrained to form (part of) another higher order object in a manner that reflects the IFO model closely.

(i) Aggregation

This construct is used to search over several object properties and then associate the properties with a higher order aggregate object.

$$r: O_r \mid \exists x_1: O_1 x_2: O_2, \ldots, x_n: O_n \langle x_1: O_1 x_2: O_2, \ldots, x_n: O_n \rangle \Theta\, r$$

The implied meaning is that free variable r ranges over objects of type O_r. For the condition to be TRUE, the additional free variables, $x_1, x_2, \ldots x_n$ together form an object of type $O_1 \otimes O_2 \otimes O_n$ which must be a sub-type of O_r, and additionally, the values associated with $x_1, x_2, \ldots x_n$ must describe a valid object of type O_r as refereed to by r.

(ii) Association

$$r: O_r \mid \exists x_1: O_1 x_2: O_1, \ldots, x_n: O_r \langle x_1: O_1 x_2: O_1, \ldots, x_n: O_1 \rangle\, \Phi\, r$$

Here, r represents a set of objects formed by association, i.e. all of the same type. Hence for the condition to be TRUE, $x_1, x_2, \ldots x_n$ must be free variables of the same type as the elements of r, and values associated with $x_1, x_2, \ldots x_n$ must form a proper subset of r, (or an element of r when $n = 1$).

2.0 RELATIONSHIPS BETWEEN FEATURES AND THEIR SPATIAL REPRESENTATIONS

Recall that there is a need to support several distinct spatial representations of a single feature, thus allowing a particular representation to be chosen according to needs defined by the current task. The resulting increase in flexibility gives rise to an entirely new problem; that of selecting or creating the most appropriate spatial representation for a given set of circumstances. Such a task clearly falls outside the experience of non-expert users. Two things are immediately obvious: (i) a means of distinguishing amongst the available candidate representations must be developed and (ii) the selection or creation of appropriate representations must be handled where possible in an automated manner and not by the user. Provision must also be made for expert users to configure and override the default actions of the selection mechanisms when they prove inadequate.

Consider the case of retrieving a single feature f of type F identified by a qualifying condition. Each f may have many distinct spatial representations forming a set

$$R = \{r_1, r_2, \cdots r_n\}.$$

Each time that f is referred to, its spatial representation must be instantiated with a single value, $r \in R$. That is, before f can be displayed, or take part in any spatial operation, a suitable representation must be found (or created), and associated with f. It is only then that any derived spatial properties pertaining to f can be computed, (such as shape, area, perimeter length). The binding of f to r is delayed until runtime to maximise flexibility according to the current context, as defined by the data under investigation, and the operation being performed.

There are a number of possible scenarios regarding different representations of each feature that must be considered.

1. A feature has exactly one candidate faithful[6] representation. This is usually the case when dealing with data supplied already in feature form, as is common with digitised boundary data. Its behaviour over time and with respect to accuracy of interpretation cannot therefore be an issue.

$$\therefore |R| = 1$$

2. f has several candidate faithful representations. Recall that there are many possible reasons why multiple representations may be required, as described in Section 1 above.

$$\therefore |R| > 1$$

The representation supplied must be selected according to pre-defined rules, as described in Section 3. Note that if none of the existing r are suitable for the current task then execution skips to **3.** below.

3. f has no suitable representations. The manifest behaviour of features of type F in image data is known to the system, but no appropriate extraction of these features has

[6]A faithful representation attempts to describe the true geometric shape of a feature at a level of detail appropriate to the current scale, as opposed to an iconic representation, that is simply used to show or mark the presence of a feature with no implied geometry [Roberts et al, 1991].

been carried out as yet. Hence any reference to the spatial properties of f must be preceded by some image processing or scene understanding method to firstly identify elements of type F, by forming a purposive interpretation of appropriate image(s) and secondly to extract f from that interpretation.

$$\therefore |R| = 0$$

As an aside, it is worth noting that in theory the number of elements in R could be infinite when interpreting image data. This in turn would cause safety of expression problems with the calculus. The situation is avoided by restricting R to include only those interpretations that can be made using the current set of methods and initialisation parameters, which is of course finite.

3.0 SELECTING AN APPROPRIATE SPATIAL REPRESENTATION

In scenario 1 above, the single available spatial representation is taken by the feature. Scenarios 2 and 3 proceed in a similar manner. In both cases, that which constitutes a good spatial representation is calculated as described below by formulating a context over which the representation would be appropriate. On completion, the required context is compared with the contexts of spatial representations already computed and stored. If a match is found, then the matching representation is returned, otherwise a representation is built specifically to satisfy the context.

Consider now the factors that determine how a particular spatial representation, suitable for a given task, is selected or created. The following example problem is used to illustrate the governing factors.

> *"Derive a series of roads from recent image data of region X to form a transportation network."*

The important constraints acting on the data are:
(i) The image used must show up roads in some way. Raw data must therefore be available in appropriate spectra, and at a high enough resolution for roads to be detectable. All of the above are properties of the sensor platform.
(ii) The data used must cover the area of interest (region X) and at the time of interest (as up to date as possible). These are properties of the images.
(iii) The interpretation of the image must emphasise linear conductivity and produce linear geometry and topology as a result, with road segments represented by arcs and nodes. (linear topology is generally used for network analysis.). These are properties of the extraction methods applied to the image data.
(iv) The resulting spatial representations of the road segments must be reconciled with any feature data already held in the database concerning other properties of these roads.

The example demonstrates that the supplying of a suitable spatial representation for features of a given type can be divided into four distinct tasks:

(i) Identify the most suitable capture device for the feature-type of interest.
(ii) Find the most suitable images that covers the area of interest at the time of interest.
(iii) Apply the most suitable methods to the image, such that the resulting representations best suit the task and are of the correct spatial type (point | line | polygon).

(iv) For each feature of interest, bind together the detected spatial descriptions with other non-spatial object properties.

The context over which a particular representation is applicable is composed of all the input variables that influence each of the tasks outlined above. A description of the tasks follows, in which each of these variables are introduced and described in turn. For each task, an accompanying IFO fragment is shown in figures 1-5.

3.1 Selecting a Capture Device (Sensor)

Since images can be derived from a number of different sources, the first task is to choose a suitable sensing platform. Two factors need to be considered:

(i) Sensing Capabilities
A capture device, such as a satellite or airborne scanner, has a set of physical properties that together determine whether it can be used to detect features of a particular type. Specifically, each platform type typically operates a number of sensors over various wavelengths. Remote sensing expertise is used to translate this information into a set of feature-types that can be detected by each particular sensor. The actual physical properties are not stored, in order to greatly simplify the task of selecting an appropriate sensor. Instead, the expert knowledge is included in the structure as a set of feature-types detectable by a sensor.

(ii) Sensing Resolution
When selecting a spatial representation it is important to consider the degree of precision with which the feature is to be described. Obviously, sensors with a higher spatial resolution will be able to produce more detailed spatial representations. To enable an automated choice to be made between various sensors as a source of images, two values are given; representing the highest and lowest map scales at which the sensor should normally be used. A decision is made by comparing these values with the map scale currently in use. By default the map scale is calculated from the resolution of the current output device (usually the screen), or set manually by the user. Table 1 shows an extract of the sensor information used for three different types of capture device.

Sensor	**Scale Start**	**Scale End**	**Detected Feature-types**
landsat_TM	1:50,000	1:1,000,000	{coastline, paddock, wood, ..., bush}
spot_panchromatic	1:25,000	1:1,000,000	{coastline, wood, road, ..., paddock}
digital_air_photo	1:10,000	1:100,000	{road, river, building, ..., lot}

Table 1: Default settings for scale ranges and preferred image sources for geographic features.

The sensor object has the following structure, as depicted in figure 1:

object_type sensor
  properties
    sensor_type: string
    set {detects: feature-type}
    start_scale: scale
    end_scale: scale

The variables required to select an appropriate sensor are:

(i) *target*: the feature_type of the subject of the current task
(ii) *scale*: from the current scale

The selection of an appropriate sensor can be expressed functionally as:

detects: $\langle sensor, target, scale \rangle \rightarrow$ boolean

applied to each sensor in turn or, given in ORC, as:

$$s: sensor | \exists\ a: \text{feature\_type}, b: \text{begin\_scale}, c: \text{end\_scale}\ \langle a, b, c \rangle\ \Theta\ s \wedge \langle target \rangle\ \Phi\ a \wedge b \leq scale \wedge c \geq scale \quad (1)$$

The result (in either case) is a set of objects of type *sensor* for which the condition is TRUE.

3.2 Selecting an image

The next task is to select a subset of images that are suitable data sources for the required spatial representation. An image typically consists of several raster layers or bands, forming a set of multi-valued pixels. Equally important are various meta-data describing where and when the image was captured and measures of uncertainty associated with capture and registration. The structure of an image object is shown in figure 2. Note that an image has a spatial window that is a specialisation of a polygon, and a capture date, that is a specialisation of a temporal object. The temporal object can represent arbitrarily complex time values such as a single date, a range of dates (jan93-dec94), or a periodic (seasonal) value across several years (jan-feb, 87-93). A good review of temporal behaviour in GIS is given by Langran, (1992).

object-type image
  properties
    image_data: raster
    platform: sensor
    coverage: spatial_window
    capture_date: date

The variables required to select an appropriate image are:

(i) s*patial_extent*
A window of interest is defined on the dataset under investigation. This is under direct control of the user and can be set to a certain area within the dataset that is currently of interest. The window gives the spatial extents of any operation and is defined by a two dimensional geometry. This geometry may be arbitrarily complex or just a simple rectangle. The default is to take the extents of the current output device.

(ii) *temporal_extent*
Analogously, the user may be concerned with a particular historical time or times and this in turn gives a temporal extent. The default here is to consider the most recent data as the most useful.

A function named *incident* is used to test for the two above conditions. It is an overloaded (polymorphic) function in that it can deal with both spatial and temporal comparisons.

incident $\langle s1, s2\rangle \rightarrow$ boolean	*s1*, *s2* are of any spatial type (point | line | polygon). Returns TRUE if *s1* and *s2* are spatially incident.
incident $\langle t1, t2\rangle \rightarrow$ boolean	temporal: *t1* and *t2* are of type temporal-object. Returns TRUE if *t1* and *t2* are temporally incident

(iii) *sensors*
From the results of the previous stage, a set of appropriate sensors was identified. This set is used here to ensure that the selected image(s) originate from suitable platforms. As a function the operation is expressed as:

covers $\langle image, \{sensors\}, spatial\_extents, temporal\_extents\rangle \rightarrow$ boolean

The function (expressed in ORC) is:

$$i\text{: image} \mid \exists\, d\text{: sensor}, e\text{: spatial\_window}, f\text{: date } \langle d, e, f\rangle\ \Theta\ i \wedge \langle d\rangle\ \Phi\ sensors \wedge \text{incident}(e, spatial\_extent) \wedge \text{incident}(f, temporal\_extent) \quad (2)$$

3.3 Selecting an Image View

An image view represents the outcome of applying a particular image interpretation method to an image. The interpretation object shown in the IFO fragment (Figure 3) is in fact an aggregation of a method and a set of instantiating parameters as described in Section 1.5. Individual instances of features are not necessarily produced, but rather the view is akin to a classified image usually containing two classes, *of_interest* and *other*, although it may be probabilistic in nature. An image view has the following structure:

object-type image_view
  properties
    source: image
    geometry: spatial_type
    formation: interpretation_method
    result: overlay

The type *overlay* is a generalisation of two forms of classified image, containing either raster or vector data, and a legend or key (Roberts et al., 1991). The image interpretation method is chosen by examining the methods listed in the appropriate object-class description. Table 2 shows a small portion of the class description for the *surfaced-road* feature-type.

sensor	interpretation method	parameters	Spatial data type
Landsat TM	linear_connectivity[7]	(expert knowledge)	line
Spot Panchromatic	sub_pixel_region[8]	(expert knowledge)	region

Table 2. Extract of the *surfaced- road* feature-type class description relating to image interpretation.

[7]The method used in this instance attempts to emphasise connectivity. Rather than applying an approach based on classification it relies on detecting edges or more accurately localised changes. A high level relaxation process is then used to link up any disjoint segments (Gahegan & Flack, 1993).
[8]The method recognises roads by a combination of spectral and edge evidence. An estimation is made of road geometry which is then overlayed onto the original image resampled to a higher resolution. The sub-sampled pixels around the road areas are reclassified accordingly (Flack et al, 1994).

An expert user can insist that a particular method be used via a mechanism described later in Section 4. In the normal case however, suitability is taken to be the appropriateness of the resulting geometry (point | line | region) to the current operation. Where different geometric types could be used to describe the feature it is important to choose a representation that suits the algorithmic requirements of the operation. The suitability of each data type to particular operations is defined in a lookup table, a sample of which is given in Table 3.

Operation	Spatial data type
calculate_area	region
calculate_centroid	region
network_connectivity	line

Table 3: Extract of the spatial data types needed for particular operations

The variables required for this task are:
(i) *images* (a set of suitable images, from the results of the previous task)
(ii) *geometry* (derived from the current operation, or NULL)
(iii) *method* (extraction method and associated parameters; set by expert, or NULL)

The function to determine if a view is suitable is therefore:

views $\langle view, \{images\}, geometry, method\rangle \rightarrow$ boolean

The ORC clause for this task is:

$$v\text{: view} \mid \exists g\text{: image}, h\text{: spatial\_type}, j\text{: interp\_method}\langle g, h, j\rangle \,\Theta\, v \wedge \langle g\rangle \,\Phi\, images \wedge (geometry = h \vee geometry = \text{NULL}) \wedge (method = j \vee method = \text{NULL}) \quad (3)$$

3.4 Applying extraction methods to Views

The goal of this stage is to reduce the set of spatial representations to those produced by a suitable extraction mechanism. A spatial representation is actually formed by applying an extraction method to an image view, thus isolating a feature of interest from the rest of the view. In simple cases, the extraction method is a region detector that groups together graphical elements or pixels into a spatial object. If the view is probabilistic, then the extraction method works by grouping at a probability threshold given as an input parameter. As with image view formation, the extraction method actually consists of a method and set of parameters. By using different parameters, it is possible for several representations of a single feature to be constructed from the same image view; hence the need to differentiate between them.

In complex scenes, the extraction methods and parameters together form sophisticated scene understanding tools utilising expert knowledge concerning feature size and shape, and other typicality measures. The methods and parameter values for extraction are stored in a similar manner to the image interpretation knowledge shown in Table 2. The *shape* of the spatial representation is described by a *spatial_object* that contains data in either raster or vector form as depicted by the IFO fragment in figure 4.

The object-type *spatial_representation* has the following structure:

object-type spatial_representation
  properties
    source: image_view
    formation: extraction_method
    shape: spatial_object

The input variables required at this stage are:
(i) *views* (a set of suitable image_views, derived from the previous task)
(ii) method (an extraction method, and associated parameters, or NULL)

The function to select only suitable spatial representations is:

extracts ⟨*representation*, {*views*}, *method*⟩ → boolean

Expressed in ORC this becomes:

$$r\text{: representation} \mid \exists k\text{: image\_view}, l\text{: extraction\_method}\langle k, l\rangle \,\Theta\, r \wedge \langle k\rangle \,\Phi\, views \wedge (method = l \wedge method = \text{NULL}) \quad (4)$$

3.5 Associating spatial representations with features

The final task is to link together a spatial representation with each feature targeted by the current operation. The structure of the *feature* object-type is shown as an IFO fragment in Figure 5 and detailed below. A discussion follows on each property of this object-type.

object-type feature
  properties
    set {description: aspatial_object}
    marker: object_marker
    tracker: tracking_method
    set {appearance: spatial_representation}

To enable the appearance of features to be tracked from one image to another, an approximate place marker is used. The *marker* must be configured initially and in most cases a centroid is sufficient. A method (*tracker*), is used to locate the correct spatial representation given a *marker*. The simple case is to look for a spatial representation whose centroid is closest to *marker*. However, the tracking function can be more complex, such as where the feature under investigation is moving (e.g. an oil slick). In this case the method used examines a specific distance around a point, possibly in a particular direction. The tracking function is applied in the following manner:

tracks ⟨*feature*, *representation*, *method*, *marker*⟩ → boolean

As spatial representations are successfully tracked, they are added to the set 'appearance' shown in the above structure.

In a similar manner to the spatial representations, other descriptive properties can and do change over time. Unlike spatial properties, they are (in between changes) fixed and not subject to uncertainty or differences in interpretation. The exception is the derived spatial properties as discussed earlier, since these are dependent on the spatial representation.

Aspatial properties together form an object that has the additional property of a temporal object over which the values have meaning. New aspatial objects are created in response to a change in some property value, and are associated into a set where each element represents a different time. The mechanism for selection of the appropriate object from the set is normally determined by the time value associated with the spatial representation, i.e. the time of image capture, and is straightforward since only one aspatial object can be valid at a given time.

Each feature has a set of spatial representations as recognised by the tracking method.

The variables used in this task are:
(i) representations (a set of spatial representations derived from the previous task)
(ii) feature (feature of interest)

In the event of retrieving a single feature, the selection of an appropriate spatial representation can be expressed as:

$$\text{represents } \langle spatial\_representation, feature \rangle: \rightarrow \text{boolean}$$

or more formally:

$$r\text{: representation } | \exists f\text{: feature, } m\text{: spatial\_ representation } f = feature \quad \langle m \rangle \; \Theta \; f \wedge \langle r \rangle \; \Phi \; m \tag{5}$$

By this stage all candidate representations are deemed to be suitable for the current task. In the event of more than one being available then selection can either be arbitrary, or further criteria may be considered.

3.6 Combining the Tasks

The ORC expressions 1 to 5 described above can be summarised as follows:

(1) detects: $\langle feature\_type, scale \rangle \rightarrow \{\text{sensor}\}$
(2) covers $\langle \{sensor\}, spatial\_extents, temporal\_extents \rangle \rightarrow \{\text{image}\}$
(3) views $\langle \{image\}, geometry, interpetation\_method \rangle \rightarrow \{\text{view}\}$
(4) extracts $\langle \{view\}, extraction\_method \rangle \rightarrow \{\text{representation}\}$
(5) represents $\langle \{representation\}.feature \rangle: \rightarrow$ representation

Combining each task, a representation is considered valid if the following compound expression is TRUE:

$$r\text{: representation} | \exists i\text{: image, } s\text{: sensor, } v\text{: view}$$
$$\text{detects}(s, target, scale) \wedge \text{covers}(s, i, spatial\_extent, temporal\_extent) \wedge$$
$$\text{views}(i, v, geometry, method) \wedge \text{extracts}(v, r, extracion\_method) \wedge$$
$$\text{represents}(r, feature)$$

For a given spatial representation the expression above evaluates to either TRUE or FALSE. Where a suitable representation cannot be found or made then one or more conditions must be removed, and where more precise control is needed then one or more conditions must be added. In practice the expert configures a series of expressions for certain scenarios. The expression given above is a useful default.

For each feature-type a list of expressions is stored in the data model. Expressions are ordered such that if one proves unsuccessful, then another (with terms relaxed or absent) is tried. The problem of assigning weights to particular terms and then choosing the outcome with the highest accumulated weight is for the present deliberately avoided. The selection of suitable weights is problematic and increases the demands placed on the expert.

3.7 A definition of Context

Context is being used here to determine which spatial representation is most suitable for the current task. Context is being applied not in a data sense (Roberts & Gahegan, 1991), but in a task oriented manner. The *context* of each representation r of a feature f of type F is defined by the expert knowledge held and:

(scale, spatial_extents, temporal_extents, geometry, interpretation_method, extraction_method)

with constants f and F. Values for each of the context terms can be gathered quite simply at the time of execution.

In the IFO diagrams shown in figures 2-4, the context is depicted for an image, image view and spatial representation by including with the each of these, the object from which they were formed. This gives a kind of audit trail, giving access to each of the variables defining the context thus allowing tests of suitability to be made. A more elegant means of expressing this idea is to imagine a *context object*, associated with the results of each stage, and storing the values for the context variables given above.

4 EXPERT CONFIGURATION AND INTERVENTION

In some cases more than one representation may be required from the same raw data. To facilitate greater control over the formation and selection of such spatial representations, the following mechanism is provided for the expert user.

Each of the context variables described are instantiated either from expert knowledge, from the current task, or from a combination of both. To produce a different spatial representation then any of these variables may be changed, or new clauses added into the expression. For example, if it was necessary to work only with the most up to date image data available, then expression (2) would become:

i: image $|\exists d$: sensor, e: spatial_ window, f: date $\langle d, e, f \rangle \Theta\ i \wedge \langle d \rangle \Phi$ *sensor* $\wedge$
incident$(e,$ *spatial_ extents*$) \wedge$
$\forall i'$: image, d' : sensor, e' : spatial_ window, f' : date $\langle d', e', f' \rangle \Theta\ i' \wedge$
$\langle d' \rangle \Phi$ *sensors* $\wedge$ incident$(e',$ *spatial_ extents*$) \wedge f' \leq f$

That is, select the image (i) where the location, time and sensor_type are suitable, and for all other suitable images (i'), none were captured at a more recent time.

4.1 Specialising Context

A particular context, when constructed for a specific purpose, is denoted by a special name so that it may be used by a non-expert. A typical subset of contexts used in agricultural landuse analysis might be:

WHEAT-MAX-EXTENTS	extraction aims at retrieving the entire field (the extraction method uses a loosely-constrained region grower and geometric evidence)
WHEAT-MIN-EXTENTS	only the cropped area is extracted (extraction is based strongly on spectral evidence)
WHEAT-FULL-CROP	where time is just before harvest (week 26 < date week > 35)
WHEAT-NO-CROP	where t is after harvest (week 40 < date < week 52)

Contexts can be logically combined (ANDed) (provided no conflict is introduced) to impose further constraints. For example:

BEST-FIELD-BOUNDARY = MAX-EXTENTS $\wedge$ FULL-CROP

BEST-CROP-BOUNDARY = MIN-EXTENTS $\wedge$ FULL-CROP

4.2 Generalising Context

In certain situations, it may not be possible to fully satisfy all constraints acting on the selection of a spatial representation. In this case a less desirable context may be applied where certain terms are relaxed or omitted. Some examples are given below:
(1) The representation chosen may fall outside of the current Time Window (T) since no appropriate image source exists for this time. Any resulting map may not accurately represent a given time.
(2) The specified extraction method fails to recognise a targeted feature.
(3) The representation selected has a scale/resolution significantly different from other representations currently selected.

Any simplified context may result in the user being unaware that certain integrities have been violated. Some simple rules are applied so that the user can be informed of potential problems. Each rule may be separately configured into 3 states (ENFORCE, WARN, IGNORE). ENFORCEd rules may not be violated. Any spatial representation breaking such a rule is never selected, even if it is the only one. WARN rules will cause a message to be printed out to a special area if violated. IGNORE rules are disabled, and no action is taken if violation occurs. The inclusion of these rules allows system behaviour to be customised to some extent for particular applications.

Rules have the general form: ⟨name, condition, state⟩, for example the rule to check for spatial representations falling outside of the currently selected temporal extent is:

TEMPORAL_CONSTRAINT: incident (date, temporal_extent) = FALSE $\rightarrow$ WARN

5 CONCLUSIONS

A description is presented of adaptive behaviour when extracting geographic features from image data under different conditions and for a range of tasks. These demonstrate the benefits to the user accrued from the architecture described above.

The system described here is an attempt to integrate the entire process of image manipulation with GIS and to propose a framework whereby the resulting semantic issues can be resolved. As such it is a step forward, but is not meant to be a definitive solution. A number of simplifications have been made along the way, in order to ensure that the resulting system is realisable. Areas where further complexity may be required include (i) the use of several image sources to produce a single spatial representation and (ii) the refinement of the range of scales over which a sensor may be used, replacing this with specific values for each feature-type. In addition, no optimisation is applied at any stage in the tasks described. In practice many standard spatial and aspatial indexing techniques could be utilised to reduce the performance degradation but these would only serve to complicate the explanations given.

Generally, the automated understanding of complex scenes is still a long way from being fully realised; any system built to address this problem will necessarily be highly complex and involved. Specifically, in order to further improve image integration two problems must be understood in greater detail. The first concerns the feature extraction process, particularly the generation of better methods for feature extraction and is the subject of ongoing research by the author and colleagues (Gahegan & Flack, 1993), (Flack et al., 1994). The second concerns the needs and goals of users of the system. These must be *fully* understood before any adaptive behaviour of a GIS can be ideally suited to the tasks which they carry out. Further research is required in this area.

REFERENCES

Abiteboul S and Hull R (1987), IFO: A Formal Semantic Database Model. ACM Transactions on Database Systems, Vol. 12, No. 3, pp 525-565.

Burrough P A (1987), Principles of geographical information systems for land resources assessment, Chapter 6, Clarendon Press, Oxford.

Davis F W and Simonett D S (1991), GIS and Remote Sensing. Geographical Information Systems, vol. 1, Ch. 14, pp 191-213, ed. Maguire D J, Goodchild M F and Rhind D W. Published by Longman, England.

Dobson J E (1993), A conceptual Framework for Integrating Remote Sensing, GIS and Geography. Photogrammic Engineering and Remote Sensing, vol. 59, no 10, pp 1491-1496.

Ehlers M, Greenlee D, Smith T and Star J (1991), Integration of Remote Sensing and GIS: Data and Data Access. Photogrammic Engineering and Remote Sensing, vol. 57, no 6, pp 669-675.

Fisher P F (1994), Probable and fuzzy models of the viewshed operation. Innovations in GIS, ed. M Worboys, Taylor & Francis, London, UK, pp 161-175.

Flack J C, Gahegan M N and West G A W (1994), The use of sub-pixel measures to improve the classification of remotely-sensed imagery of agricultural land. Proc. Seventh Australasian Remote Sensing Conference, pp 534-543, Melbourne, Australia.

Gahegan M N and Flack J C (1993). Query-centred interpretation of remotely sensed images within a GIS. Proc European Conference on Geographic Information Systems, Genoa, Italy, March 1993, pp 942-949. Published by EGIS foundation, PO Box 80.115, 3508 TC Utrecht, The Netherlands.

Gahegan M N (1994), A consistent user-model for a GIS incorporating remotely-sensed data. Innovations in GIS, ed. M Worboys, Taylor & Francis, London, UK, pp. 65-74.

Gahegan M N and Flack J C (1994). A model to support the integration of image understanding techniques within a GIS. Under review.

Glenn D, Compton B and Crosbie P (1992), Dynamic integration of spatially referenced data independent of data source format. Intergraph Corporation: Huntsville (Alabama).

Grosky I W and Mehrotra R (1992). Image Database Management. Advances in Computers, vol. 36, pp 237-292.

Hunter G and Goodchild M F (1993). Communication of uncertainty in spatial databases. Proc. AURISA '93, Adelaide Australia. AURISA Inc., ACT.

Korth H F and Silberschatz A (1991), Database System Concepts (second edition), Chapter 1, McGraw-Hill.

Lacroix M and Pirotte A, (1977). Domain-Oriented Relational Languages, Proc. 3rd Int. Conf. on Very Large Databases (VLDB).

Langran G, (1992), Time in geographic information systems. Technical issues in geographic information systems, Taylor and Francis.

Roberts S A and Gahegan M N, 1991, Supporting the notion of context within a database environment for intelligent reporting and query optimisation. European Journal of Information Systems, vol. 1, no 1, pp 13-22.

Roberts S A, Gahegan M N, Hogg J and Hoyle B S, 1991, Application of object-oriented databases to geographic information systems. Information and software technology, vol. 33, no. 1, pp 38-46.

Taylor A, Cross A, Hogg D C and Mason D C, 1986, Knowledge-based interpretation of remotely sensed images. Image and vision computing, vol. 4, no 2, pp 67-83.

Ullman J D, 1988. Database and knowledge-base systems, vol. 1. chapter 3. Computer science press, Maryland USA.

Wang, F, 1988, A knowledge-based system for highway network extraction. IEEE Transactions on Geoscience and Remote-Sensing, vol. 26 no. 5.

Worboys M F, Hearnshaw H M and Maguire D J (1990), Object-oriented data modelling for spatial databases. International Journal of GIS, vol. 4, no 4, pp 369-383.

Worboys M F, Hearnshaw H M, Maguire D J, 1991, Object-oriented data and query modelling for geographical information systems. Proc. 4th International Symposium on Spatial Data Handling, Ed. Brassel K & Kishimoto H, Dept. of Geography, University of Zurich, Switzerland, vol. 2, pp 679-688.

FIGURES

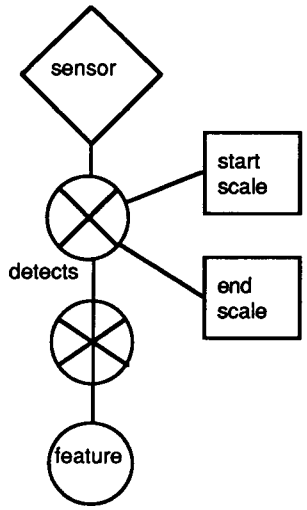


Figure 1: IFO fragment describing the structure of the SENSOR object.

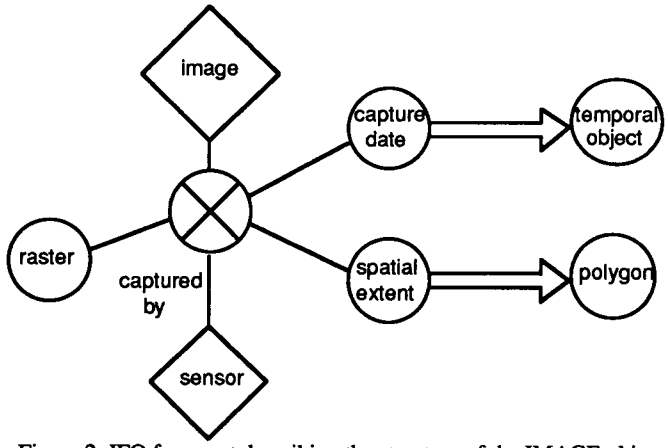


Figure 2: IFO fragment describing the structure of the IMAGE object.

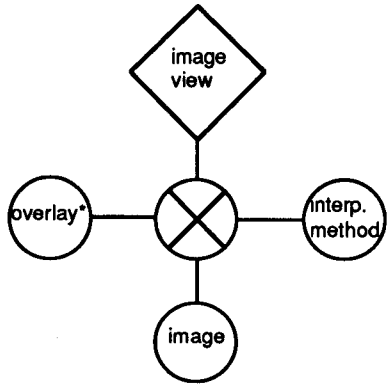


Figure 3: IFO fragment describing the structure of the IMAGE_VIEW object.

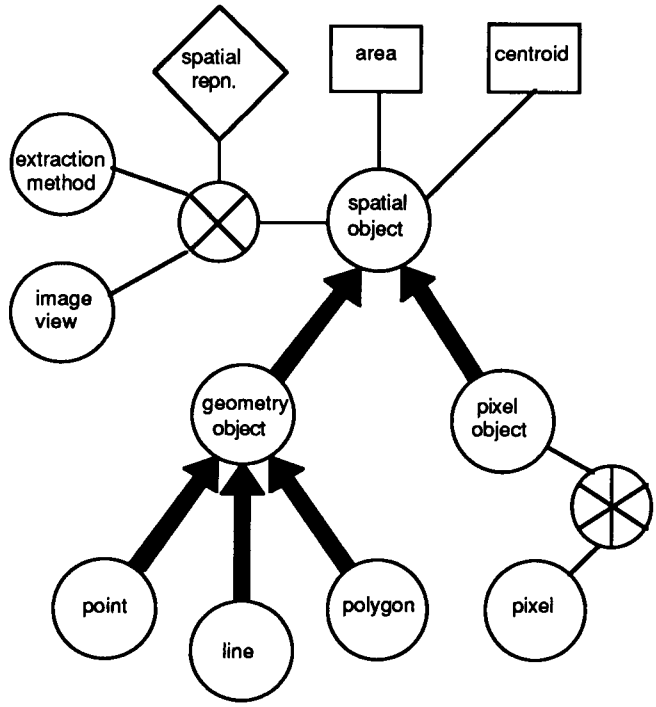


Figure 4: IFO fragment describing the structure of the SPATIAL_REPRESENTATION object

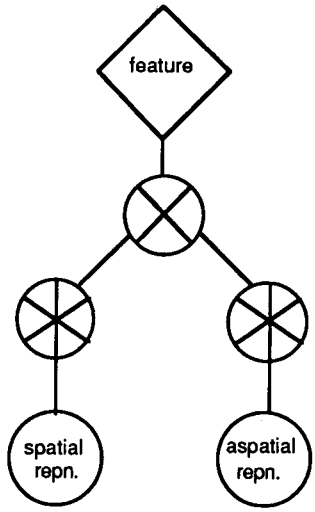


Figure 5: IFO fragment describing the structure of the FEATURE object.

Spatial Querying

Spatial Access Methods and Query Processing in the Object-Oriented GIS GODOT[1]

Volker Gaede
Institut für Wirtschaftsinformatik, Humboldt Universität zu Berlin,
Spandauer Str. 1, 10178 Berlin, Germany
gaede@wiwi.hu-berlin.de

Wolf-Fritz Riekert
FAW Ulm, Helmholtzstr. 16,
89081 Ulm, Germany
riekert@faw.uni-ulm.de

ABSTRACT. In this paper, we describe the spatial access method *z-ordering* and its application in the context of the research project GODOT, which is based on the commercial object-oriented database system ObjectStore [LLOW91]. After identifying a range of spatial predicates, we show that the intersection join is of crucial importance for spatial joins. Next, we propose an efficient method for query processing, which takes advantage of z-ordering and uses the conventional indexing mechanisms offered in current database systems (e.g., relational and object-oriented).

1. INTRODUCTION

Spatial data management is an important database application area where object-oriented database concepts can be utilized efficiently in a variety of ways [GL94]. As a result, such techniques are gradually being integrated into GIS (geographic information system) products, and object-oriented spatial database systems are being implemented on top of commercial OODB systems [GR93, SV92].
A major conceptual problem that arises in this context concerns the definition of the relevant spatial operations and their efficient computation. Many efforts have been undertaken to define a continuum of spatial queries, also known as *query space*, which is complete in a certain sense, and to define data structures which support the efficient computation of these queries (cf. [Ege94, Mol92]). One of the most important spatial operations is the *spatial join*. Günther [Gü93] gives the following

[1]The GODOT project has been conducted at FAW (Research Institute for Applied Knowledge Processing) in Ulm, Germany since 1992. GODOT was commissioned by the State of Baden-Württemberg, Siemens Nixdorf Informationssysteme AG and Siemens AG.

definition of spatial joins:
The spatial join of two relations R and S, denoted by $R \bowtie_{i\theta j} S$, is the set of tuples from $R \times S$ where the ith column of R and the jth column of S are of some spatial type, θ is some spatial predicate, and $R.i$ stands in relation θ to $S.j$. According to the terminology used in geographic information systems and object-oriented databases we will also refer to the tuples $r \in R$, $s \in S$, etc. as *spatial objects* or *geo-objects* and to the relations R, S, etc. as *object extents* or *sets of objects*. We assume (without loss of generality) that one dedicated column in the relations R, S, etc., is of some spatial data type and represents the *geometry* of the related spatial objects. We will therefore write $R \bowtie_\theta S$ as a shorthand for the spatial join $R \bowtie_{i\theta j} S$ whenever i and j denote these dedicated columns in the relations R and S, respectively.
A brief survey of the literature yields a wide variety of spatial predicates that may be used for spatial joins, including

- intersects(r, s)
- contains(r, s)
- is_enclosed_by(r, s)
- distance(r, s)ΘE, with $\Theta \in \{=, \leq, <, \geq, >\}$ and E a given real number
- northwest(r, s)
- adjacent(r, s)

These predicates check if some spatial relationship between two objects is satisfied. Another class of spatial operations which will become important in this context concerns the "transformation" of geo-objects. Examples are ($r \in R, s \in S$): convex_hull(r), buffer(r, d), northwest(r), geometric_union(r, s), geometric_difference(r, s), etc. A closer inspection of these spatial predicates and operations shows that the *intersection* join $R \bowtie_{\text{intersects(r,s)}} S$ plays a crucial role for the computation of all the other predicates.
For predicates such as contains, encloses, or adjacent, the intersection join is an efficient filter that yields a set of tuples (r, s), typically much smaller than the Cartesian product $R \times S$, which still contains all solutions to the original query. That is, the intersection join is an efficient *preprocessing* step.
Most of the remaining predicates (for example, distance, northwest) can be computed by using the intersection join as a *postprocessing* step as follows. In order to compute the spatial join $R \bowtie_{\theta(r,s)} S$ for one of those θ-predicates, we can apply some function $\phi_\theta : R \to R'$ and $\psi_\theta : S \to S'$ to R and S respectively, then compute the intersection join $R' \bowtie_{\text{intersects(r,s)}} S'$. If the functions ϕ_θ and ψ_θ have been properly defined, a tuple (r, s) belongs to the spatial join $R \bowtie_{\theta(r,s)} S$ if and only if $R' \bowtie_{\text{intersects(r,s)}} S'$. For example, it is possible to compute the northwest query $R \bowtie_{\text{northwest(r,s)}} S$ (asking for all pairs (r, s), where s is situated in the northwest of r) in the following way[2]:

$$\text{northwest}(R) \bowtie_{\text{intersects(r,s)}} S$$

[2]Let the northwest-function for a relation in the following formula be defined as the element-wise application of the northwest-function on the elements of the relation. That is, by using the spatial attribute of the given tuple, the northwest function precomputes the area located to the northwest.

That is, instead of checking the northwest predicate for every pair of tuples in the Cartesian product of both relations, the northwest *function* is applied to the first relation involved, followed by an intersection join with the second relation
With this concept it is possible to reduce the most common spatial joins to an ordinary intersection join, which requires some pre- and/or postprocessing. The following table gives a concise overview of the required transformations.

Postprocessing

1.	$R \bowtie_{\mathsf{is\_enclosed\_by(r,s)}} S$	$\rightarrow$	$\sigma_{\mathsf{is\_enclosed\_by(r,s)}}(R \bowtie_{\mathsf{intersects(r,s)}} S)$
2.	$R \bowtie_{\mathsf{contains(r,s)}} S$	$\rightarrow$	$\sigma_{\mathsf{contains(r,s)}}(R \bowtie_{\mathsf{intersects(r,s)}} S)$
3.	$R \bowtie_{\mathsf{adjacent(r,s)}} S$	$\rightarrow$	$\sigma_{\mathsf{common\_border(r,s) \wedge not(overlap(r,s))}}$ $(R \bowtie_{\mathsf{intersects(r,s)}} S)$
4.	$R \bowtie_{\mathsf{overlay(r,s)}} S$	$\rightarrow$	$\mu_{\mathsf{compute\_intersection(r,s)}}(R \bowtie_{\mathsf{intersects(r,s)}} S)$

Preprocessing

5.	$R \bowtie_{\mathsf{distance(r,s)}<d} S$	$\rightarrow$	$\mathsf{buffer(R, d)} \bowtie_{\mathsf{intersects(r,s)}} S$
6.	$R \bowtie_{\mathsf{northwest(r,s)}} S$	$\rightarrow$	$\mathsf{northwest(R)} \bowtie_{\mathsf{intersects(r,s)}} S$
7.	$R \bowtie_{\mathsf{nearest\_neighbor(r,s)}} S$	$\rightarrow$	$\mathsf{min(buffer(R, d))} \bowtie_{\mathsf{intersects(r,s)}} S$

In the above table σ denotes the well-known selection operator and μ is the tuple constructor operator. It should be pointed out that the above transformations are independent of the actual representation of the geometry.
The remainder of this paper is organized as follows. In Section 2 we give a short introduction to z-ordering. In Section 3 we develop a more formal description of the underlying z-value calculus and section 4 shows that it can be used for processing and optimizing the most common spatial queries by using a minimal set of essentially four ϕ-functions. In Section 5, we illustrate the overall architecture and show how the technique of query rewriting can be applied to achieve the presented concepts.

2. Z-ORDERING

Although there is a large number of spatial access indexing techniques [Gü88, Kol90], most of these techniques cannot be easily integrated into a commercial database system. This paper, however, shows that the *z-ordering* scheme (see, for example, [OM88]) has the potential of being implemented on top of a commercial database system. One reason for this is that the z-ordering is a logical access method and not a physical one, even if physical clustering is beneficial and can be exploited by ObjectStore.
Z-ordering assumes a recursive decomposition of the plane into a hierarchical configuration of rectangular areas known as *z-regions*. These z-regions can be identified canonically by binary code strings known as *z-values* consisting of ones ('1') and zeros ('0'). If the z-value of a given region prefixes an other z-zvalue, then the former region encloses the latter: for example, 00 encloses 001. Figure 1 shows some z-regions and their associated z-values. Any spatial object can be approximated by a set of z-regions which form the lowest upper bound to the spatial extension of the

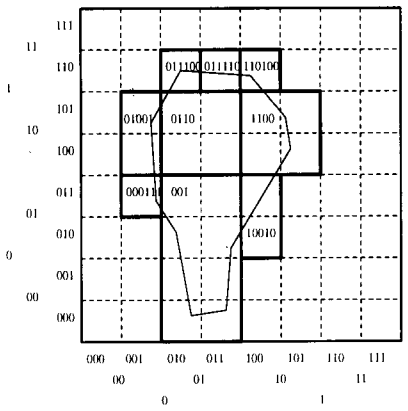


Figure 1: Decomposition of a given object using the z-value approximation.

object.[3]
Since these z-regions are uniquely identified by their associated z-values, the spatial extension of any spatial object can be approximated by a set of z-values.

$$Z = \{z_1, z_2, ..., z_n\} \quad (n \text{ finite})$$

The basic idea in the work presented here is to compute results of spatial queries (particularly of the intersection join) on the basis of z-value sets. As we will show in the following section, the computation of spatial queries using z-values can be performed by means of simple (boolean) operations rather than by computationally expensive geometric operations. Like most spatial access methods, however, the result of a query using z-values is only a set of candidate objects, which must be postprocessed using the exact geometries, to eventually find those objects that really satisfy the inquiry.

3. FORMAL MODEL OF QUERY PROCESSING USING Z-ORDERING

There are two possible views to describe a spatial database using z-ordering:

1. The z-value centered view: each existing z-value is bound to one or more objects.

2. The object centered view: each object is composed of a set of z-values which are stored as a set-valued attribute of the object.

Note that these two perspectives are totally different from the viewpoint of query processing. In the first case, a query using z-values would scan the z-value extent (a collection of all existing z-values) for matching z-values and then search objects using these z-values. In the second case, one has to retrieve first the objects of the (geo-) object extent (Ω) and then scan their extent for matching z-values. For our discussion we will use the latter model as a basis.
In the remainder, we will call the set of all (true) prefixes of the given z-value the

[3]As can be seen from this introduction, the z-ordering scheme can easily be extended to handle n-dimensional objects.

upper hull and all the z-values which are prefixed by a given z-value, the *lower hull.* Note that the region R corresponding to the z-value set Z_R is enclosed by all regions corresponding to z-values in the upper hull. Conversely, R *encloses* all regions corresponding to z-values in the lower hull.
In order to develop a more formal model of query processing using z-values, we introduce the following terminology.

- z_i: a sequence of values from $\{0, 1\}$
- $|z_i|$: length of z_i, i.e., the number of digits in the $0, 1$- sequence
- Z: a non-empty set of z-values
- $g = max(|z_i|)$: a measure for the granularity of z-values in the database
- $\mu_i(z_k)$: cutting operator, cuts the given z-value to the length i, i.e., leading digits are preserved up to the length i
- $\epsilon_i(z_k)$: extension operator, generates all possible extensions up to a given length i
- $\underline{Z} = \{\epsilon_k(z_i) || z_i| \leq k \leq g \ \& \ z_i \in Z\}$: lower hull
- $\overline{Z} = \{\mu_k(z_i) | 1 \leq k \leq |z_i| \ \& \ z_i \in Z\}$: upper hull
- $Z^+ = \underline{Z} \cup \overline{Z}$: hull closure
- Ω: extent of existing (geo-) objects
- $o_i.Z$: z-value set of object o_i

To evaluate the submitted queries, we have to define how they can be computed in the domain of z-values. For this purpose, we introduce a minimal set of essentially four different ϕ functions. A more precise definition of the ϕ-function is given in [GG94]. For our purpose it suffices to know, that each of ϕ-functions ϕ_c transforms a set of z-values Z into another set of z-values Z'. More formally:

$$\phi_c : Z \xrightarrow{c} Z'$$

where c denotes a certain constraint defined on this function.

1. $\phi_{\text{buffer(d)}}(Z)$: generate, for a given set Z, a z-value buffer of distance d; for convenience we assume that distances with respect to z-regions are measured in units of the grid size of the underlying cell structure
2. $\phi_{\text{adjacent}}(Z) = \phi_{\text{buffer(1)}}(Z)$: generate, for a given set of z-values Z, all possible values of adjacent cells (the distance 1 stands for the grid quantum)
3. $\phi_\gamma(Z)$: $\gamma \in \{\text{north}, \text{south}, \text{east}, \text{west}\}$: generate all z-values situated in the direction γ.
4. $\underline{\phi}(Z) = \underline{Z}$: compute the lower hull

5. $\overline{\phi}(Z) = \overline{Z}$: compute the upper hull

6. $\phi^+(Z) = \underline{Z} \cup \overline{Z}$: carry out the hull closure for the given set; for brevity, will often write Z^+ instead of $\phi^+(Z)$

It should be pointed out that these functions are defined on the level of z-values and not on the exact object geometries. Hence, all operations can be carried out very fast.
Examples:

1. $\underline{\phi}$: given $Z = \{001, 01\} \Longrightarrow \underline{\phi}(Z) = \{001*, 01*\}$,
   that is, generate all z-values having the same prefix[4]

2. $\overline{\phi}$: given $Z = \{001, 01\} \Longrightarrow \overline{\phi}(Z) = \{001, 01, 00, 0\}$,
   that is, apply the cutting operator μ_i successively to the given set Z.

3. $\phi_{\text{buffer(d)}}$: The effect of the buffer operator is depicted in figure 2.

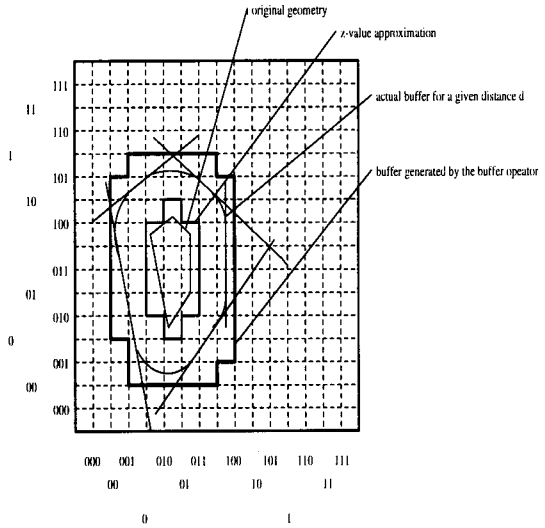


Figure 2: For the given geo-object and its z-value extension a buffer of distance d is build. Both, the exact buffer and its z-value buffer a depicted. Note, however, that the z-value buffer is build on basis of the original z-value set.

After introducing these fundamental functions, we can define the most important operations and equivalences to be used in the sequel:

1. intersection:

$$\begin{aligned} \textsf{intersects}(Z', Z'') &\iff \underline{Z'} \cap \underline{Z''} \neq \emptyset \\ &\iff \{\exists z' \in Z' : z' = \mu_{|z'|}(z'') \;\&\; z'' \in Z''\} \vee \\ &\quad\; \{\exists z'' \in Z'' : z'' = \mu_{|z''|}(z') \;\&\; z' \in Z'\} \end{aligned}$$

[4]We will elaborate on how to represent these values in terms of intervals in section 4.1.

$$\iff \exists z' \in Z' : z' \in Z''^{+} \iff Z' \cap Z''^{+} \neq \emptyset$$
$$\iff \exists z'' \in Z'' : z'' \in Z'^{+} \iff Z'' \cap Z'^{+} \neq \emptyset$$

For efficiency reasons, the equivalence of the last two lines very important, since the complexity depends on the cardinality of the participating sets $Z'(Z'')$. Due to the above equivalences, the spatial intersection (for z-values) can be reduced to a simple enclosure test for one given value at best.

2. is_enclosed_by:

$$\begin{aligned} \mathsf{is\_enclosed\_by}(Z', Z'') &\iff Z' \subseteq \underline{Z''} \\ &\iff \forall z' \in Z' : z' \in \underline{Z''} \end{aligned}$$

In the above equation all elements are tested for enclosure in $\underline{Z''}$. A slight variation of the above query reveals a way how this test can be done in a more efficient way.

$$\nexists z' \in Z' : z' \notin \underline{Z''}$$

Instead of testing all elements of z' for enclosure in $\underline{Z''}$, the negation of the above equation can be tested more efficiently, i.e., it is sufficient to find one element of Z' not in $\underline{Z''}$ to negate the assumption of enclosure. Informally stated:

$$\neg\mathsf{is\_enclosed\_by}(Z', Z'') \iff \exists z' \in Z' : z' \notin \underline{Z''}$$

3. contains: $\underline{Z'} \supseteq Z''$

$$\mathsf{contains}(Z', Z'') = \mathsf{is\_enclosed\_by}(Z'', Z')$$

This equivalence enables the optimizer to reduce the **contains** test to an is _enclosed_by test or vice versa.

4. QUERIES IN DETAIL

In the next few sections, we will argue different kinds of queries and discuss their implementation.

1. Point Query
   Given a point p, find all objects enclosing this point.
   The given point p is represented by its z-value z_p. Therefore the query asking for all objects containing this point would be

$$Q_{\text{point}} = \{o | z_p \in \underline{o.Z} \ \& \ o \in \Omega\}$$

Processing this query would be inefficient since for each object o in the database one would have to compute the lower hull. A more efficient way is:

$$Q_{\text{point}} = \{o | \exists z \in \overline{z_p} : z \in o.Z \ \& \ o \in \Omega\}$$

This means that it suffices to find one z-value in the object extent which is included in the upper hull of the z_p.

2. Window Query
   Given a window w, find all objects o intersecting this window. The given window w will be represented by its z-values Z_w. This is in general a set of z-values, since the given window is only congruent in exceptional cases with the underlying grid structure. To find all objects, which are enclosed by this window, we have to carry out a hull closure, i.e., generate all interesting z-values.

$$\begin{aligned} Q_{\text{window}} &= \{o | Z_w^+ \cap o.Z \neq \emptyset \ \& \ o \in \Omega\} \\ &= \{o | \exists z \in Z_w^+ : z \in o.Z \ \& \ o \in \Omega\} \end{aligned}$$

3. Region Query
   Given a region of arbitrary shape r, find all objects intersecting this region. In contrast to the window query, where the search region has rectangular shape, the region query encompasses more general shapes. However, from the view of query processing using z-values the region query can be handled analogously to the window query, i.e., for a given region r and its corresponding set of z-values Z_r, the query could be expressed in the following way:

$$\begin{aligned} Q_{\text{region}} &= \{o | Z_r^+ \cap o.Z \neq \emptyset \ \& \ o \in \Omega\} \\ &= \{o | \exists z \in Z_r^+ : z \in o.Z \ \& \ o \in \Omega\} \end{aligned}$$

4. Intersection Query
   Given an object o_k (or a set), find all pairs of objects (o, o_k) intersecting each other:

$$Q_{\text{intersection}} = \{(o, o_k) | \exists z \in o_k.Z : z \in \underline{o.Z} \ \& \ o \in \Omega\}$$

   Note that is not possible to express such a query using the ObjectStore query interface, since ObjectStore does not offer a tuple constructor. A more relaxed version of the above query which can be answered using the ObjectStore query facility would be

$$Q_{\text{intersection}} = \{o | \exists z \in o.Z : z \in (o_k.Z)^+ \ \& \ o \in \Omega\}$$

5. Enclosure Query
   Given an object o_k, find all objects enclosing this object completely.

$$Q_{\text{enclosure}} = \{o | \forall z \in o_k.Z : z \in \underline{o.Z} \ \& \ o \in \Omega\}$$

   This query would be again inefficient since for all (geo-) objects of extent Ω the lower hull would be computed. More efficient variations are:

$$Q_{\text{enclosure}} = \{o | \forall z \in o_k.Z, \exists i \geq |z| : \mu_i(z) \in o.Z \ \& \ o \in \Omega\}$$

   and

$$Q_{\text{enclosure}} = \{o | \forall z \in o_k.Z, \exists o.z \in \overline{z} \ \& \ o \in \Omega\}$$

6. Containment Query
   Given an object o_k, find all objects enclosed by o_k.

$$Q_{\text{containment}} = \{o | \forall z \in o.Z : z \in \underline{o_k.Z} \ \& \ o \in \Omega\}$$

7. Distance Query
   Given an object o_k, find all objects within a distance d to o_k.

$$Q_{\text{distance}} = \{o | \exists z \in (\phi_{\text{buffer(d)}}(o_k.Z))^+ : z \in o.Z \ \& \ o \in \Omega\}$$

8. Neighborhood-Query
   Find all neighbors of a given object o_k.

$$Q_{\text{neighbor}} = \{o | \exists z \in (\phi_{\text{buffer(1)}}(o_k.Z))^+ : z \in o.Z \ \& \ o \in \Omega\}$$

9. Northwest-Query
   Find all objects situated in the northwest direction of a given object o_k. This query can be expressed by combining two of the introduced ϕ-functions.

$$Q_{\text{northwest}} = \{o | \exists z \in ((\phi_{\text{north}}\phi_{\text{west}})(o_k.Z))^+ : z \in o.Z \ \& \ o \in \Omega\}$$

10. Nearest (Farthest) Object(s)
    This kind of query can only be expressed in an unsatisfactory way, since one has to find first the nearest (farthest) distance before the hull closure can be conducted.

$$Q_{\text{nearest\_object}} = \{o | \exists z \in (\phi_{\text{min(buffer(d))}}(o_k.Z))^+ : z \in o.Z \ \& \ o \in \Omega\}$$

$$Q_{\text{farthest\_object}} = \{o | \exists z \in (\phi_{\text{max(buffer(d))}}(o_k.Z))^+ : z \in o.Z \ \& \ o \in \Omega\}$$

As the reader may have noticed, the first three queries could have been subsumed under the intersection query, although the given formulation of the intersection query is geared toward existing objects.

4.1. Representation of Z-values

To handle the z-values in the context of the database systems efficiently, one has to find a data structure that can be manipulated easily and efficiently. To do this, we found that transforming the tuple (z-value, z-value-length) to an unsigned integer is the most promising way. One possible transformation is

$$z' = b \cdot n + l$$

with

- z': new z-value;
- b: a zero-padded binary representation of the z-value;

- n: arbitrary number, which has to be at least greater than a certain minimum;
- l: number of valid digits, i.e., the z-value length.

The above formula transforms the tuple (z-value, z-value-length) to one unique `integer`-value. This transformation is not dense, but it is topology preserving (in terms of z-ordering), which means that region queries can be expressed in the most natural way.
For example, for the granularity of 6 the z-value `001` is padded to `001000 = 8`. The length of the given z-values is 3 and thus one gets $z' = 8 \cdot n + 3$ for some given n. By choosing n greater than a certain number, it is possible to extract easily the length of the z-value. To guarantee a fast transformation, n should preferably be chosen to the basis 2.

5. QUERY REWRITING

5.1. ObjectStore Query Interface

The syntax of ObjectStore queries[5] is quite simple if one assumes only (very) simple operations. The general structure of a query is `any_extent[: int_expression :]`, whereby we assume in the sequel, that each class has an extent consisting of all objects of this specific class. This query returns all the objects satisfying the `int_expression`, i.e., the `int_expression` is unequal to 0. Existential queries can be expressed by enclosing the "where clause" into `[% %]` brackets. This query returns only one object of the extent satisfying the given predicate. If none is found, an exception is raised.
It should be mentioned that it is not possible to express a join between two extents in a declarative fashion. Possible work arounds prohibit query optimization. A more detailed description of the query interface and query optimization can be found in [OHMS92].

5.2. Optimizing Queries Containing Spatial Predicates

Optimizing spatial queries is somewhat different to "conventional" query optimization since queries frequently consist of a spatial and a non-spatial (thematic) part whereby the spatial part normally includes expensive predicates in terms of computational cost. Our query interface, which is built on top of the ObjectStore query interface, elaborates only on the spatial part, that is, rewrites and optimizes the spatial part using z-values. This rewritten query is submitted together with the non-spatial part to the ObjectStore query optimizer. The overall architecture is depicted in figure 3. Generally, all possible queries containing spatial predicates and taking advantage of z-ordering have to undergo the following transition:

$$\text{GEO-QUERY}(...) \rightarrow \text{z-value-condition(s)} + \text{geo-query}(...)$$

[5]We will restrict our explanations, however, to the program interface since they apply to the ad-hoc interface with some restrictions too.

This statement illustrates that the original query is transformed to an augmented query, whereby the first selection is performed using the z-value condition (filter step), generated in the process of rewriting the query. All objects which have passed this z-value test successfully form a set of candidates, which has to be postprocessed by applying the remaining predicates. Postprocessing is necessary since the final evaluation must be done using the exact geometries.
As can be seen from figure 3 it is necessary to hold some additional information to evaluate the submitted query. To avoid, for example, the superfluous generation of not existing z-values while applying the ϕ-functions, it is beneficial to keep track of existing z-values. One further needs the ObjectStore query interface or the meta-object protocol to acquire further information.

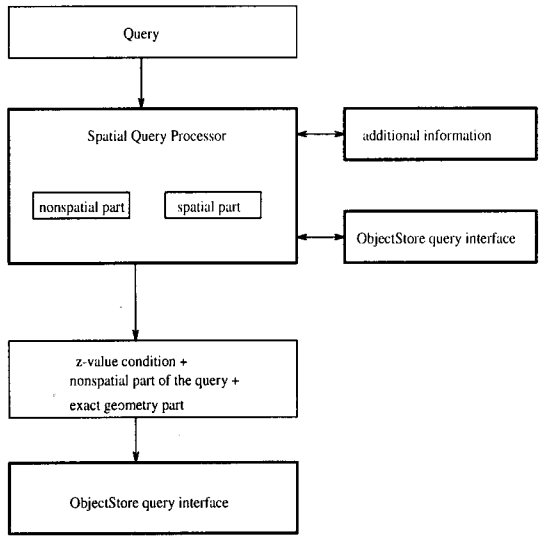


Figure 3: Architecture of the Query Processor

5.3. Example: Intersection Query

Assume that we are interested in finding the place where the television tower in East-Berlin (`tvteb`) is located. The tower is represented in our example by the z-value `''1000''` [6] (extended object, i.e., not point). Let us further assume that the maximum length (granularity) of the z-values is six ($g = 6$) and that there exists an extent of z-values, whereby each z-value can occur multiple times and each existing z-value is at least associated with one object.
Hence, the intersection query would be expressed in the following way:

```
Extended_Object * tvteb;
extent[: INTERSECTION(tvteb) :];
```

By using

$$Q_{\text{intersection}} = \{o | \exists z \in o.Z : z \in (o_k.Z)^+ \ \& \ o \in \Omega\}$$

[6] We give the z-values using their `0,1`-representation since we found it more instructive.

the query is tranformed to [7]

```
extent[: (zvalue == "1000*" || zvalue == "100" || zvalue == "10" ||
        zvalue == "1") && intersection(geometry, tvteb->geometry) :];
```

Assuming the transformation given in section 4.1, the intersection query can be further transformed to ($n = 5$) intervals and integers. The final final query would be:

```
extent[: (161 >= zvalue && zvalue <= 181)
                && intersection(geometry, tvteb->geometry) :];
```

This query returns the the place we are looking for: the Alexanderplatz.
As can be seen from the above query it is not necessary for all elements to test explicitly if they are enclosed in the (discrete) set of z-values. The transformation given above enables one to regroup the z-value set and identify intervals. From the viewpoint of query processing, this test can be performed much more faster. It is further noteworthy, that this query can be processed in at least an index-supported manner by creating indices on z-values.

6. CONCLUSION AND FUTURE WORK

In this paper we have shown that the technique of z-ordering is general enough to be applied in combination with (object-oriented) database systems. One gets the most out of the offered database functionality without substantial loss of performance compared to proprietary systems.
It should be noted that the z-ordering approach described in this paper does not rely on precomputed relationships between spatial objects. Whenever a new spatial object is entered into the database, it is not necessary to determine and to establish topological relations to all its neighboring objects. It is sufficient to compute the associated z-value set. Existing objects may be disregarded while inserting a new spatial object into the database. In particular, this makes it easy to compute spatial joins between objects coming from different sources.
Furthermore, it has been demonstrated that the concept of the ϕ-function can be applied in this setting to enable the optimization and processing of user-defined functions in at least an index-supported way. Hence, the combination of z-ordering and ϕ-function seems very useful, since one can use spatial indexing and user-defined functions to compute spatial joins in a straightforward way with a (hopefully) substantial performance gain. Although we can not report performance results at the moment, first tests are promising.
Current and future work on the optimizer includes the integration of not yet implemented query functionality and further optimization.

ACKNOWLEDGEMENT

We gratefully acknowledge the interesting discussions with Oliver Günther and his careful revision of drafts of this paper.

[7] Writing ''1000*'' is not allowed in ObjectStore and is only used here to express the prefix feature of the z-values, i.e., all intersecting z-regions share the same prefix. The star is used as a wildcard for zero or more digits following the prefixing sequence of digits.

REFERENCES

[BKS93] Thomas Brinkhoff, Hans-Peter Kriegel, and Ralf Schneider. Comparison of approximations of complex objects used for approximation-based query processing in spatial database systems. In *Proc. 9th Int. Conf. on Data Engineering*, pages 40–49, 1993.

[Ege94] Max J. Egenhofer. Spatial SQL: A query and presentation language. *IEEE Transactions of Knowledge and Data Engineering*, 6(1), February 1994.

[GG94] Volker Gaede and Oliver Günther. Processing joins with user-defined functions. Technical Report 94-013, ICSI, Berkeley, California, March 1994.

[GL94] Oliver Günther and Johannes Lamberts. Object-oriented techniques for the management of geographic and environmental data. *The computer journal*, 37(1), 1994.

[GR93] Oliver Günther and Wolf-Fritz Riekert. The design of GODOT: An object-oriented geographic information system. In *IEEE Data Engineering Bulletin*, September 1993.

[Gü88] Oliver Günther. *Efficient Structures for Geometric Data Management.* Springer-Verlag, 1988.

[Gü93] Oliver Günther. Efficient computation of spatial joins. In *Proc. 9th Int. Conf. on Data Engineering*, 1993.

[Kol90] Curt P. Kolovson. *Indexing Techniques for Multi-Dimensional Spatial Data and Historical Data in Database Management Systems.* PhD thesis, University of California at Berkeley, 1990.

[LLOW91] Charles Lamb, G. Landis, Jack Orenstein, and D. Weinreb. The ObjectStore database system. *Communications of the ACM*, 10(34), October 1991.

[Mol92] Martien Molenaar. Formal data structures and query spaces. In *Konzeption und Einsatz von Umweltinformationssystemen.* Springer-Verlag, 1992.

[OHMS92] Jack Orenstein, Sam Haradhvala, Benson Margulies, and Don Sakahara. Query processing in the ObjectStore database system. In *Proc. of the 1993 ACM Int. Conf. on the Management of Data. SIGMOD Record*, volume 22, pages 403–412, June 1992.

[OM88] Jack A. Orenstein and Frank Manola. Probe: Spatial data modeling and query processing in an image database application. *IEEE Transactions on Software Engineering*, 14:611–629, May 1988.

[SV92] Michel Scholl and Agnès Voisard. Object oriented database system for geographic applications: An experiment with O_2. In *The O_2 BOOK*, pages 585–618. Morgan Kaufmann, San Mateo, California, 1992.

QUERY CLASSIFICATION, A FIRST STEP TOWARDS A GRAPHICAL INTERACTION LANGUAGE

VINCENT SCHENKELAARS[0]
Erasmus University / Tinbergen Institute
Faculty of Economics, Department Computer Science
P.O. Box 1738, 3000 DR Rotterdam
E-mail: Schenkelaars@cs.few.eur.nl
Phone: +3110-4081416 Telefax: +3110-4526177

ABSTRACT. Earlier research was mainly focused on the operation side of as gis system. Less attention has been paid to the users view on GIS. This paper describes an Interaction Language which bridge the gap between the system developers and the end users view on GIS. First the user queries are classified. After that, the connection between operations and query classes is made. For each query class, an Abstract Interaction Tool (AIT) needs to be developed. A short desciption is given about the principles of the AIT concept.

1. INTRODUCTION

In almost all currently available commercial Geographical Information Systems an SQL-like query language is used to query a spatial database. It is noticed that these query languages are not suitable to be used directly by end-users. It is argued that the query language is not suitable to be used as a spatial query language at all [6]. There are several extensions proposed on SQL: Spatial SQL [5, 7], GEOQL [9], SQLSX [4]. All these dialects will lead to an SQL3 standard [3]. Although this standard provides the necessary geometric types and functionality, it still contains three severe deficiencies.

[0]This research was funded by a grant from TNO-FEL under AIO-contract number 93/399/FEL

First, there is no way of defining the graphical output. SQL is a query language, not an Interaction Language. The query ouput in a GIS is often used to create a map. Being able to specify the graphical appearance of the query results, makes a GIS more usefull. Second, SQL is incapable to process qualitative queries, knowledge queries, and metadata queries. These are fundamental problems arising from the relational framework of SQL. A GIS user might need this type of queries. Finally, SQL3 is complex. The spatial operations do not fit well into the relational model. SQL3 is a *database* query language not a GIS query language. Even if the SQL syntax is expressed in a graphical way by building a query tree, as is currently done in GEO++ [14], it still is a database language. Object-Oriented approaches like in [8] do not leave the database level either.

GIS has one important feature which distinguishes it from other database applications: the map. It seems to be a waste not to use the map during interaction. Users are often familiar with paper maps and have knowledge about what is presented on the map. They know how to read a map. Using the map for interaction gives the user a well-known context reference.

The research goals are to design a Graphical User Interaction Language which is user-friendly, logical, extendible, and uses the map as an interactive object which can be interrogated. This language will be modeled as a layer on top of a database query language, although there seems to be no obstruction to use the language to interact directly with a spatial database. The language consists of two parts: A query language for dealing with the user queries, and a presentation definition language, which defines the graphical presentation of the results.

Before an attempt can be made to develop an Interaction Language for GIS and other spatial information systems, an investigation of the user demands on spatial operations should be performed. Not only the question "What does a user want to do?" but also the question "How does the user want to perform his task?" should be answered. This will be investigated in the following sections.

2. GIS OPERATIONS CLASSES

There are a large number of distinct operations a GIS-user could want to perform. A complete set of these operation is not yet defined. There are,

however, a small number of fundamental operations which can be used as building bricks for an Interaction Language. These fundamental operations can be classified in the following way [1, 11, 12, 13]. *Selection* operations retrieve data from the database; This is the most common performed operation. *Transformation* operations transform selected data for scale, orientation, projection, and presentation; *Reclassification* operations assign new attribute values to a set of objects based on initial attribute values, geometry, or topological relations; *Geometric Computation* operations calculate simple geometric values like area, length, perimeter, and distance, but also more complex calculations like buffer zone, and average distance; *Transitive Closure Analyses*[8] calculate shortest path, travel time, and maximum flow. This can be done on a line graph or a polygon network; *Neighbourhood* operations use neighbourhood information to calculate slope, diversity, profile, and interpolation; *Map Overlay* combines two or more maps; Both spatial and thematic attribute values can be combined; *Statistics* operations calculate statistical characteristics like percentage, distribution, frequency, and correlation of objects of different classes over a specified area.

3. QUERY CLASSES

Beside operation classes one can do a classification on the queries as well. A user does not think in terms of distinct operations, but in terms of queries and answers. It is therefore useful to try to classify user queries. Possible user queries have been gathered and decomposed. Query components can be distinguished by examining the type and number of input and output parameters of the operation. The components are ordered in the following class division:

- *Thematic Queries:* Asking for objects with a simple, thematic attribute. The input parameter is a one dimensional attribute of an object. Examples are: "On which crossroads did happen an accident last month?", "Which buildings have more than six stories?" and "Does Amsterdam have more than 500,000 citizens?". Note that the last example only results in a yes or no answer, while the other two would result in a set of objects.
- *Geometric Queries:* Asking for objects with a certain geometric property. The input parameter of an instance of this queryclass can be a

geometric attribute of an object like in "Which cities have an area larger than $80km^2$?". Note that the geometric attribut of this example is not the area but the polygon describing the shape of the city. The input parameter can also be a geometric relation with an other object like in "Which buildings are within 10 km distance of this chemical plant?". Also directional queries can be stated with instances from this query class like in "In which direction is the railroad station?". This query results in a compass point answer and is an example of a query in which a virtual point (a users position) is used as a reference point. Different would be the question "Where is the railroad station?", which results in a single absolute location.

- *Topological Queries:* Asking for objects which have a certain topological property or have a certain topological relation with another object or an object type. Examples: "Which lakes have islands?"; "Does the A4 lead to Amsterdam?"; "What cities does the A4 cross"; "Which highways cross a forest?". The first example asks for an object with a certain topological property. The next three examples contain relations between respectively two objects, an object and an object type, and two object types.
- *Transitive Closure Queries:* Asking network related questions. The network can be a graph or a polygon map. The input parameters of this type of queries are a network a (set of) starting point(s), and a (set of) target point(s). The result of the query is a (set of) path(s) formed by a sequence of members of the network. Examples: "What is the shortest path from Amsterdam to Paris?", "What routes can I take with a sailing ship, having a draught of 1.20 meter, to go from Sneek to Stavoren?", and "What is the nearest gas station?".
- *Spatial Analysis and Deductive Queries:* Asking questions on special analytic problems. The common property of the queries from this class is that the input parameter is a map made of objects of different types. The output can be anything from a single location like in "What is the best place to put an air-defense unit in order to control an area as large as possible, but at least controlling this hill?", a set of objects like in "What airbases are in range the next two hours?", an area like in "Which area will be affected by this fire considering this wind speed?", or a complete new map like in map overlay. Usually complex calculation has to be performed in order to get an answer. Other examples are

surface interpolation and slope calculation in a TIN [10].

- *Knowledge and Metadata Queries:* Asking questions about the reasoning process or the data model. Examples are: "What is this object?", "What are the possible soil classifications?", and "Why are these objects called neighbours?".

With the standard boolean operators **NOT**, **AND**, and **OR**, all possible user queries can be constructed.

The query classes form a users view on GIS. The operation classes on the other hand are a system developers view on GIS. Relating these two classes will lead to a more user friendly GIS without sacrificing any expressive power. Below an attempt is made to map the operation classes onto the query classes. If this mapping is succesful, it is proved that the set of query classes is as powerful as the set of operation classes. In other words, all operations can be performed by combining instances of some query classes. This means that lifting the abstraction level from operations to queries does not restrict the user. Unfortunately, as is shown below, this is not totally possible. But with the addition of a Presentation Language, the abstrction is possible. This Presentation Language enables the user to specify the way query results procesed. Results can be presented in tables or graphically with a map.

Some operation classes can be mapped onto the query classes with relative ease. Members of the *Selection* class are retrieval queries and are straight forward mapped into one of the query classes. *Reclassification* operations are updates on the database. These can be seen as attribute updates and are therefore mapped into the thematic queries. Operations from the *Calculation* class are part of a query, e.g. "Which cities have an area larger than 80km^2?" contains the *area* operation. *Transitive Closure* maps trivial on its query class counter part. *Neighbourhood*, *Statistics*, and *Map Overlay* are examples of spatial analyses queries.

The *Metadata and Knowledge* queries are addressed in [6] and come into play when a user asks "What does this symbol mean?". The answer could then be: "This is an airfield". When a GIS has a kind of reasoning mechanism, knowledge queries like: "Why are these two objects called neighbours?" could be stated. Since it is not yet common to have a GIS combined with a knowledge system, this type of operations did not come into view often. An Interaction Language however should have the possibility to state this kind of queries.

Transformation operations are performed on results of queries. These operation are not covered by any query class. Transformation is clearly a part of the Presentation Language. This proves that a query language at itself is not powerful enough to perform all the requested operations. The combination of a Presentation Language and a Query Language provides the requested expresive power.

4. ABSTRACT INTERACTION TOOLS

We now have a framework to develop an Interaction Language for Geographical Information Systems. The "What?" part of the Interaction Language has been described. The "How?" part is still unknown. For each query class an Abstract Interaction Tool (AIT) [2] needs to be designed. Only a general overview of the AIT principals is given, more details can be found in [2]. The AIT approach will lead to a uniform way of defining a user query for all query instances in a class. Syntactically it makes no difference whether the user asks for the perimeter or the area of an object. The AIT model offers a tree-like hierarchy of interaction objects. Each AIT represents a subtree and can be considered as an abstract input device containing a syntax-like expression of the required input-pattern. At the higher hierarchical levels, an AIT describes the functions required by a user (what), while the lower level AITs describe the way they are accomplished (how). The AIT level handles only the user interaction. Links with an application are needed to get a real working system.

4.1. AIT Syntax

An AIT is an input-driven module. It can be viewed as an object containing data structures and methods which are triggered by physical or symbolic input. Syntactically, the definition of an AIT contains up to seven sections:

TOOL *Name = input expression*
formal parameter declaration
local function and variable declarations
local AIT definitions
INIT
PROMPT

ACTION
END *Name*

The most important part of an AIT is the input expression. This expression is mandatory. It is a regular expression which specifies on or more legal input strings. the input strings are matched against physical or symbolical input tokens. The source of the input tokens are the AITs names in the input string. One could write an AIT for a mouse. This AIT will handle all evernts generated by the mouse. The following operations (in descending priority) are used: ' \ $ | * ; + & ? <>. Evaluation of the input expression is from left to right, but parentheses may be used to override the default ordering. Let A, B, C, and D be arbitrary operands, that is AITs or AIT expressions, the operators are defined as follows:

Expr	Name	Meaning
A ; B	sequence	A followed by B
A + B	selection	A or B
A & B	concurent	A and B are processed parallel or concurent
A ? B	significance	significant part of A followed by B [1]
<A> ; B	option	optional A followed by B
n'A	exponent	A;A;...;A with n equal to length of string
A * B	Kleen star	Zero or more time A followed by B
A $	postfix	repeat A until A returns false
A | B | C	predicate	if A then B otherwise C [2]

As an example a simple AIT describing the picking of an object from a map is shown below. The AIT returns the object identification number `oid` of the picked object. The objects are stored in a database. A mouse movement while the left mouse button is being pressed cancels the pick action. First there can be an arbitrary number of mouse movements before the left mouse button is pressed. After that, the object at the cursor location is being searched for in the database. If the object is found, it is highlighted. The AITs *MouseMove*, *LeftMouseDown*, and *LeftMouseUp* are all defined elsewhere.

TOOL *PickObject = MouseMove(x,y)* ; LeftMouseDown(x,y) ;*

[1] This is only meaningfull if A is an AIT expression. e.g. A = a;b;c.
[2] C may be omitted

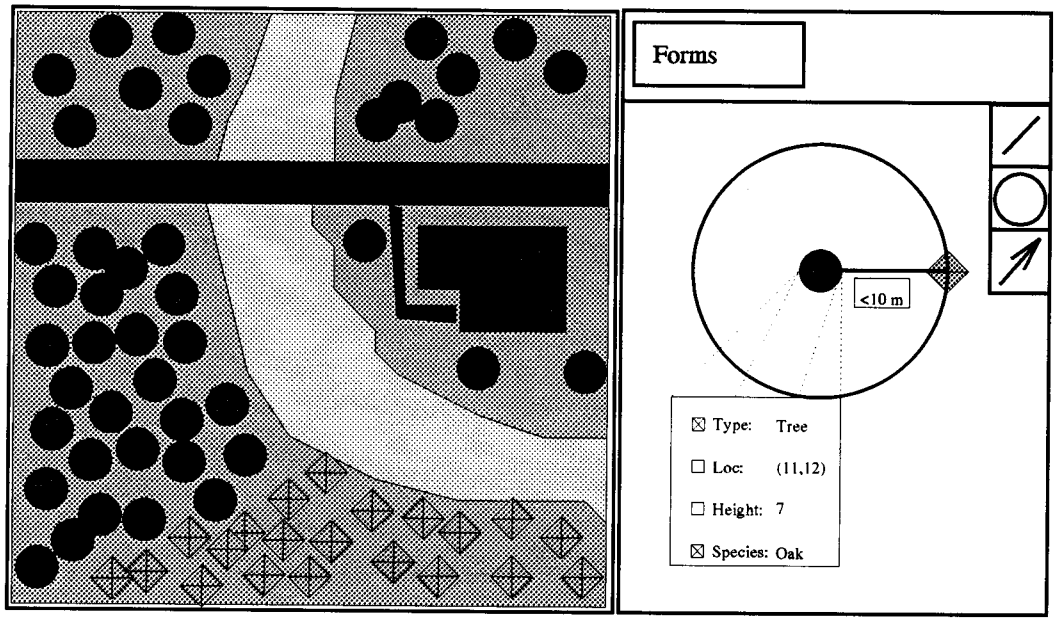


Figure 1: Forrest Scenery

```
                    (MouseMove(x,y)* + LeftMouseUp())
    PARAM OUT oidtype oid;
    VARS int x,y;
    PROCS oidtype findobject(int, int);
    /* No local AIT definitions */
    /* No Init */
    PROMPT ON Message("Searching for object");
    PROMPT OFF Message("Picked object %d", oid);
    ACTION oid = findobject(x,y);
           Highlight(oid);
           PickObject = true;
END
```

In the ideal situation, an AIT is a black box of which only two things need to be known: What function does it perform, and what input pattern triggers it.

In the example *MouseMove* and *LeftMouseDown* are active at the same

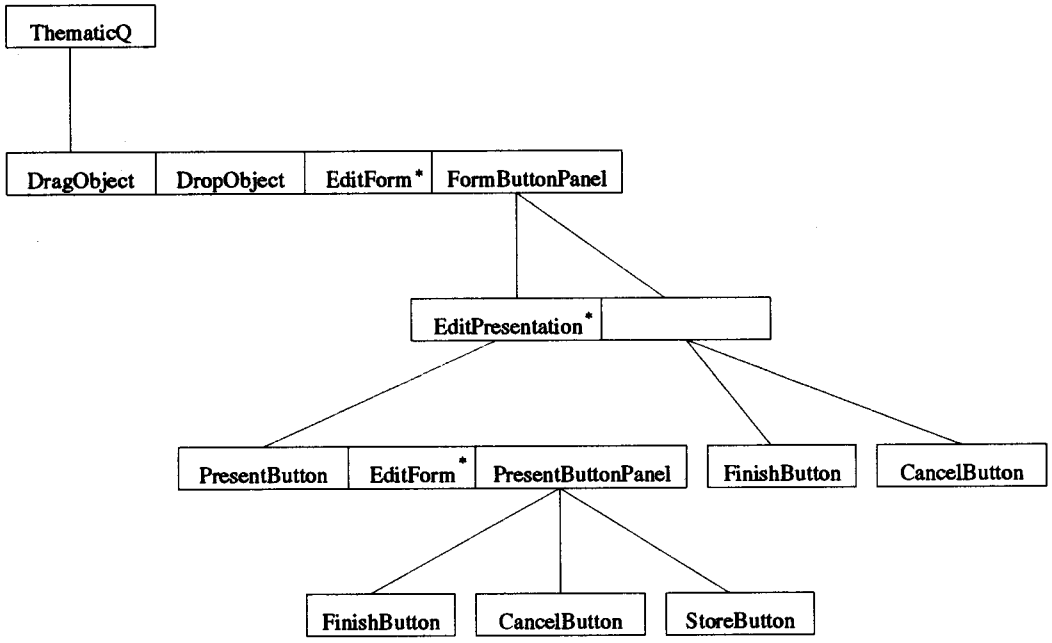


Figure 2: AIT tree of a thematic Query

time. If *LeftMouseDown* becomes true (when the user presses the left mouse button, *MouseMove* and *LeftMouseDown* are both deactivated and *LeftMouseUp* and a new *MouseMove* are triggered. If either of these two becomes true, PickObject becomes true and its actions are being executed.

4.2. AIT Tree Structure

As stated earlier, a collection of matched AITs form a hierarchical parse tree. This tree is being traversed in a breaddth first search. All leaf AITs are activated and ready to trigger their parent AIT whenever a user event takes place. The parsing of the AIT tree is explained using an example.

Figure 1 shows a simple GIS interaction window. This is not an implemented application but only a design. It contains a scenery with Oaks (circles) and Pine trees. On the right side is a box with label "Forms". This box is called the *forms slot.* This slot is used to perform thematic queries. When a user drags a copy of an object from the map into the forms slot, a form is popped up. The form has for each attribute a line made of a tag box,

the attribute name, and the attribute value. When the box is tagged, the attribute value is used as a selection criterion in the thematic query. When the box is not tagged, the attribute is not used in the query. The form contains further a Name field, a "Presentation Button", a "Cancel" button, and a "Finished" button.

A thematic query can be constructed as follows: Drag an object (preferably one with the requested properties) into the forms slot. Tag the attributes which are needed in the query. Edit the values field if necessary. Change the presentation if the default presentation does not suffice. Finally, press the "Finished" Button. Now the database query (e.g. a SQL query) can be constructedi by the systemi and send to the database system for further processing. The input pattern of the AIT tree looks like:

TOOL *ThematicQ* = *DragObject* ; *DropObject* ; *EditForm** ;
FormButtonPanel
END *ThematicQ*

TOOL *FormButtonPanel* = *EditPresentation** ;
(*FinishButton* + *CancelButton*)
END *FormButtonPanel*

TOOL *EditPresentation* = *PresentButton* ; *EditForm** ;
PresentButtonPanel
END *EditPresentation*

TOOL *PresentButtonPanel* = *FinishButton* + *CancelButton* + *StoreButton*
END *PresentButtonPanel*

For clarity, the formal parameters of the AITs are omitted. The input patterns form a grammar. Figure 2 shows the tree structure of this example. The example consist of two databasxe queries. The first retrieves the attributes of the object droped into the forms slot. The second is a kind of Query-By-Example [15]

Below the forms slot in figure 1 is a graphical edit area. In this area, a user can construct a query by dragging objects from the map and add operators. In the example the query "Which oaks are closer than 10 meters to a pine tree?"

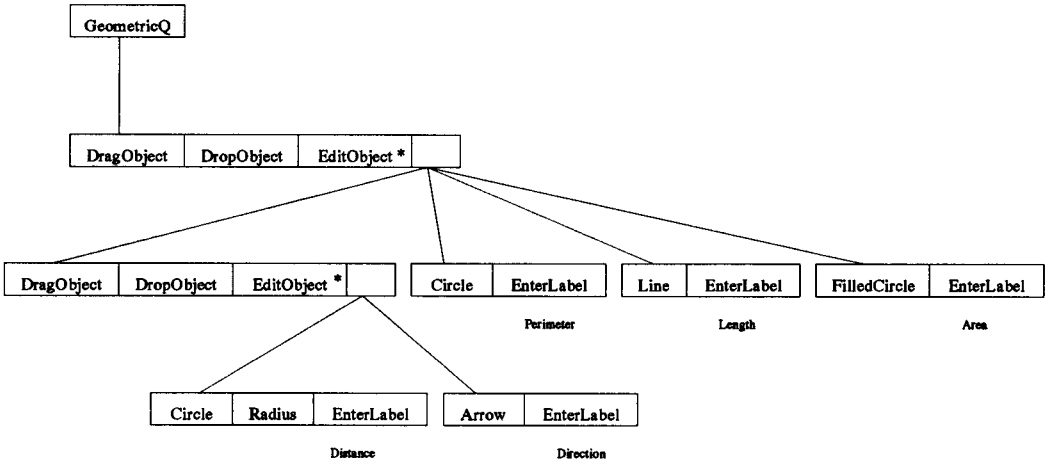


Figure 3: AIT tree of a geometric Query

The construction of a geometric query is depicted in figure 3. The figure is a simplified tree. It has five distinct paths which lead to the query examples: *Distance*, *Direction*, *Perimeter*, *Length*, and *Area*. The order in which the user adds the objects to the query construction area is important. The first object is always the target object. This object is the one that meets the query requirements. The other objects are the criteria. In the example only objects can be selected, but by auditioning new paths in the tree, new interaction methods can be introduced. For example the query "In what direction is the railroad station?" can not be execute with the example tree. When the path *Arrow ; DragObject ; DropObject ; DragObject ; DropObject* is added, this query can be constructed.

5. CONCLUSIONS & DISCUSSION

A number of authors have already identified the basic operations that should be available in a GIS. Little attention has been paid to how these operations are offered to the user. An attempt is made to approach this topic from the users view. Users think more in terms of questions than in terms of operations. A set of query classes is developed and mapped onto the operation classes. This mapping was necessary in order to prove that the expresive power of both classes are equal. It turned out that query classes are not

powerful enough by them selfs. Some other language is needed for presentation issues. The task of this language is to allow the user to specify the presentation of the query results. What color to use, what transformations are needed, what symbols to use, are all aspects of this language. This Language still needs some research.

For all query classes a AIT has to be developed for perfomring the queries. The AITs are handling all user interaction. A problem remains how to show the user what possible action he can take at any situation. A advanced help system should be a part of the AIT description.

The presented interaction system is still in a design phase. Only two of the query classes are (partly) developed now. Future implementation and testing are needed.

ACKNOWLEDGEMENTS

This research was funded by a grant from TNO-FEL under AIO-contract number 93/399/FEL. Comments on early version of this paper made by Aske Plaat, Roel van der Goot, and Mark Polman were very helpful. They are thanked for doing this job. Peter van Oosterom is acknowledged for his advise and supervision of this research.

REFERENCES

[1] Joseph K. Berry. Fundamental operations in computer-assisted map analysis. *International Journal of Geographical Information Systems*, 1(2):119–136, 1987.

[2] Jan van den Bos. Abstract interaction tools: A language for user interface management systems. *ACM Transactions on Programming Languages and Systems*, 10(2):215–247, April 1988.

[3] Michael Bundock and Jonathan Raper. Towards A standard for spatial extensions for SQL. In *Proceedings of EGIS '92*, volume 1, pages 287–304, Munich, 1992. EGIS Foundation.

[4] M.S. Bundock. Sql-sx: Spatial extended sql - becoming a reality. In *Proceedings of EGIS '91*, Brussels, 1991. EGIS Foundation.

[5] Max J. Egenhofer. An extended sql syntax to treat spatial objects. In *Proceedings of the 2nd International Seminar on Trends and Concerns of Spatial Sciences*, New Brunswick, 1987.

[6] Max J. Egenhofer. Why not SQL! *International Journal of Geographical Information Systems*, 6(2):71–85, 1992.

[7] Max J. Egenhofer and Robert D. Franzosa. Point-set topological spatial relations. *International Journal of Geographical Information Systems*, 5(2):161–174, 1991.

[8] Michel Mainguenaud and Marie-Aude Portier. Cigales: A grapical query language for geographical information systems. In *Proceedings of the 4th International Symposium on Spatial Data Handling*, volume 1, pages 393–404, Zürich, 1990. IGU Commission of GIS.

[9] Beng Chin Ooi. *Efficient Query Processing in Geograpic Information Systems*, volume 471 of *Lecture Notes in Computer Science*. Springer-Verlag, 1990.

[10] Thomas K. Peucker and Nicholas Chrisman. Cartographic data structures. *American Cartographer*, 2(1):55–69, 1975.

[11] T. R. Smith, S. Menon, J. L. Star, and J. E. Estes. Requirements and principles for the implementation and construction of large-scale Geographic Information Systems. *International Journal of Geographical Information Systems*, 1(1):13–31, 1987.

[12] P. Svensson and H. Zhexue. Geo-SAL: A query language for spatial data analysis. In *2nd Symposium, SSD '91: Advances in Spatial Databases*, volume 525, pages 119–142. Springer-Verlag, Zürich, 1991. Conference Proceedings.

[13] C.D. Tomlin. *Geograpic Information Systems and Cartographic Modeling*. Prentice Hall, Englewood Cliffs NJ, 1990.

[14] Tom Vijlbrief and P. van Oosterom. The GEO++ system: an extensible GIS. In *Proceedings of the 5th International Symposium on Spatial Data Handling*, volume 1, pages 40–50, Charleston, 1992. IGU Commission of GIS.

[15] M.M. Zloof. Query-by-example: A database language. *IBM Systems Journal*, 16(4):324–343, 1977.

3D Modelling

A COMPUTATIONAL MODEL OF THE BASIC MEANINGS OF GRADED COMPOSITE SPATIAL RELATIONS IN 3-D SPACE

KLAUS-PETER GAPP
Cognitive Science Program, Dept. of Computer Science
Universität des Saarlandes, D-66041 Saarbrücken, Germany

Email: gapp@cs.uni-sb.de
Phone: (+49 681) 302-3496
Fax: (+49 681) 302-4421

ABSTRACT. Natural language, a primary human communication medium, facilitates human-machine interaction and could help towards a more flexible use of Geographic Information Systems. In this context, spatial relations serve as the connecting link in the translation process between visual and verbal space. However, in certain situations, elementary topological or projective spatial relations are not enough to locate an object exactly. We have, therefore, extended to composite relations our existing computational model of elementary spatial relations. The compositions are obtained by using the scaled minimum of the elementary relations.

1. INTRODUCTION

The prepositions in their spatial meanings (cf. [Retz-Schmidt 88]) combined with descriptions of placing, an *object to be localized* (*LO*) and a *reference object* (*REFO*), build the class of *localization expressions* (cf. [Herskovits 86]). The semantic analysis of spatial prepositions leads to the term *spatial relation* as a meaning concept independent of target-language. Conversely, the definition and representation of the semantics of spatial relations is an essential condition for the synthesis of spatial reference expressions in natural language. For computing topological relations fundamental concepts of algebraic topology and set theory can be used. The approach of Egenhofer (cf. [Egenhofer & Herring 90, Egenhofer & Franzosa 91]) is one of the best known in this domain. More general approaches, which include also projective spatial relationships, are specifying conditions for object configurations, such as the distance between objects or the relative position of objects, to compute spatial relations (cf. [Hanßmann 80, Carsten & Janson 85, Peuquet 87, Gapp 94a, Olivier et al. 94]. In this sense, a spatial relation characterizes a class of object configurations.

The work described here is part of the *VITRA* (VIsual TRAnslator) project which deals with the relationship between vision and natural language (cf. [André et al. 89]).

Before we focus on the computation of composite spatial relations, the computational model of the basic meanings of elementary spatial relations in 3-D space is introduced (cf. [Gapp 94a]).

Classes of Spatial Relations					
topological		projective		exception	
German	English	German	English	German	English
an	*at*	*hinter*	*behind*	*zwischen*	*between*
bei	*near*	*links*	*left*		
fern	*far*	*neben*	*beside*		
in	*in*	*rechts*	*right*		
nahe	*close to*	*über*	*above*		
		unter	*below*		
		vor	*in front of*		

Table 1: German spatial relations and their English counterparts

2. COMPUTING THE ELEMENTARY SPATIAL RELATIONS

Following [Landau & Jackendoff 93], people do not account for every detail of the objects involved when they apply spatial relations. These authors propose that spatial relations depend mainly on boundedness, surface, or volumetric nature of an object and its axial structure (cf. also [Talmy 83]). Hence, an *approximate* algorithm can be utilized for the computation of spatial relations, considering only the essential shape properties of an object (cf. [Mukerjee & Joe 90, Abella & Kender 93, Rajagopalan 94]). For instance, in most cases the center of gravity is sufficient to approximate the located object, because its position is the only information that counts for establishing spatial relations. The following idealizations are currently provided in our model:

- *Center of gravity*
- *Bounding rectangle* (*BR*)
  The bounding rectangle of an object with respect to a direction vector $\vec{v}$, is the minimum rectangle which is aligned to $\vec{v}$ and contains the 2-D representation of the object.
- *2-D representation*
  The base of each object (necessary when perceiving objects from a bird eye's view, e.g., maps).
- *Bounding right parallelepiped* (*BRP*)
- *3-D representation*
  The complete geometrical description of an object.

Three distinct classes of static spatial relations are considered: The topological relations (e.g., *at* and *near*), the projective relations (e.g., *in front of*, *right*, and *below*), and the relation *between*, which takes an exceptional position in the group of spatial relations.[1] Table 1 shows the 13 different German spatial relations and their English counterparts considered at the moment.

[1]In the sequel only the English expressions for the German prepositions are used. Slight differences between the German and the English may appear.

The algorithm for the computation of the basic meanings of the topological relations (like *at* or *near*) considers the distance between the *LO* and the *REFO*. The distance is measured in a *local extended coordinate system* which is aligned to the *REFO*'s orientation. The extended axes are scaled by the *REFO*'s extension in each of the three dimensions (Figure 1).

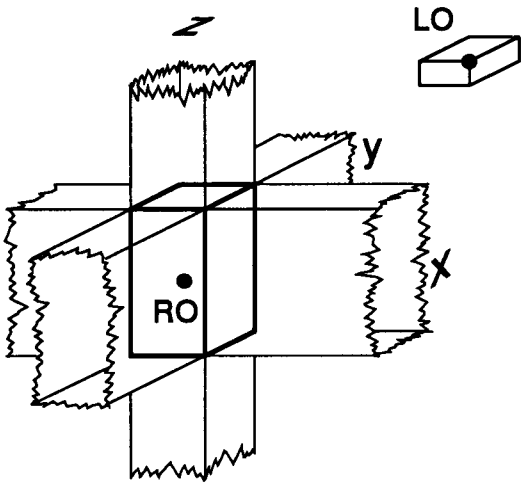


Figure 1: Definition of the *local coordinate system*

The computation of projective relations also has to include the scaled (*local*) deviation angle of the *LO* from the canonical direction implied by the relation.[2] The dependency of the local distance and the local angle from the *REFO*'s extension ensures for a bigger *REFO* in a dynamic enlargement of a relation's region of applicability (cf. [Gapp 94a]). Spatial relations are fuzzy (cf. [Freeman 75, Logan 94]) and the evaluation of the applicability of a spatial relation can differ from one person to another (cf. [Kochen 74]). We map, therefore, the local distance and the local angle to user adjustable *cubic spline functions* $Spline_{Rel}$, which allow also a cognitively plausible continuous gradation of the applicability of the relation (cf. [Zadeh 65, Zadeh 68]). Figure 2 shows examples of spline definitions for the relations *at* and *near*. The product of both spline values determines the *degree of applicability* DA_{Rel} of an elementary projective spatial relation.

To cope with relations like *in front of* or *right*, in three dimensions some approaches project the objects to a plane orthogonal to the direction of gravitational force (cf. [Olivier et al. 94]). But this is insufficient, for instance, if a plane is flying over a house, such a procedure would describe the plane as being *in front of* the house, if the plane's x- and y-coordinates are pointing to a location in front of the house. Therefore it is necessary to cope with a fully 3-dimensional computational model.

[2]This holds in any case for questions using *where*, e.g., "*Where is the next phone booth?*". It is likely that this kind of distance concept has to be dropped, if the query is fully specified, i.e., the two objects and the relation are used in the query. For Instance, if an object *A* is exactly to the right of an object *B* but far away: The question, "Is *A* to the right of *B*" must be affirmed, independently of the distance. To get a well-founded answer we designed (3D-SEE), an experimental environment to get psychological evidence about these problems. (3D-SEE) will also allow us to adjust our computational model to the average as humans do.

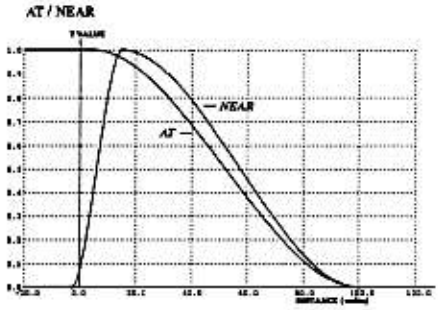


Figure 2: Evaluation functions of the proximal region concerning *at* and *near*

The model we presented can be applied to 2-dimensional, e.g., maps, as well as to 3-dimensional environments. In the latter case, we get a 3-dimensional extended region of applicability for spatial relations. This seems reasonable and necessary when we look at Figure 3, because *"the LO in front of the REFO"* is an adequate application of the spatial relation *in front of* although there is a vertical difference between the *LO* and the *REFO* (cf. [Gapp & Maaß 94]).

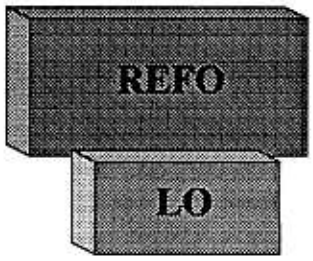


Figure 3: The *LO in front of* the *REFO*

To get an idea of a spatial relation's applicability structure we extended the so-called *potential field* presentation of a spatial relation (cf. [Yamada et al. 88, Schirra & Stopp 93]) into the third dimension (cf. [Gapp 94b]). For instance, Figure 4 is a visual example of the 3-D applicability structure of *above* (a) and *right* (b) using a building as *REFO*. The darker the grey areas of the relation's region of applicability are, the higher is the degree of applicability.

3. COMPOSITIONAL USE OF SPATIAL RELATIONS

Where necessary to localize objects more precisely, German usually uses more than one spatial relation. Normally not more than two relations are combined, e.g., *the LO is in front of and to the right of the REFO*. The use of compositions is very common in German, because they are direct combinations of the simple relation's terms: For instance, "*rechts vor*" means the same as "*to the right and in front of*" in English. Thus to get a composite localization expression using two prepositions, one has only to combine the elementary expressions appropriately.

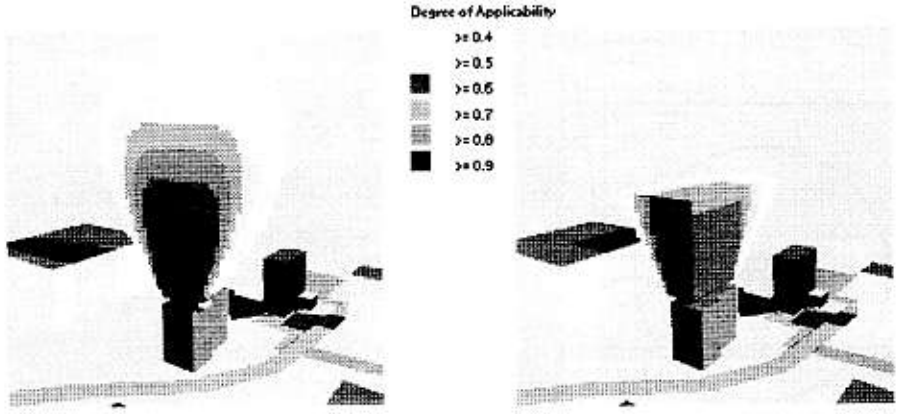


Figure 4: Cross-sections of the 3-D applicability structure of *above* and *right*

Spatial prepositions	*at* *near*	*close to*	*in front of* *behind*	*right* *left*	*above* *below*
at,near					
close to			X		
in front of,behind					
right,left	X	X	X		X
above,below					

Table 2: Ordered 2-place compositions of German spatial prepositions

But not every elementary preposition can be combined with another. There is a limited number of compositions allowed, depending on the kind of the preposition and its position in the composite expression. The relations we take into account are the topological *at*, *near*, and *close to* and the projective *in front of*, *behind*, *right*, *left*, *above*, and *below*. Table 2 gives an overview of the possible ordered two-place compositions of the spatial prepositions (in German use).[3]

According to the table, the prepositions *right* and *left* occupy the first position and *at*, *near*, *far*, *in front of*, *behind*, *above*, and *below* occupy the second position in the composition. Only *close to* can occupy both positions depending on the other preposition.

4. COMPUTATION OF COMPOSITE SPATIAL RELATIONS

It should be possible to define a semantics for each possible combination of elementary spatial relations. But it wouldn't make sense, because in the language only a restricted set is used as we have showed above. We consider, therefore, the compositions given in

[3]Note: It is also possible to have some more compositions, e.g., the ordered composition of *in front of* and *at* which can be expressed as "*vorne an*" in German. But "*vorne*" is an adverb and not a preposition and the basic meaning of *vorne an* is not exactly the combination of the elementary prepositions' basic meanings.

table 2.

We define two classes of 2-place composite spatial relations, the *2-place composite projective* (Rel^{2p}_{cp}), a composition of two elementary projective relations, and the *2-place composite projective topological* (Rel^{2p}_{cpt}), which is a combination of a projective and a topological.

Before the algorithms are explicitly given, we will first consider the requirements for a computational model. To give an idea of how different models of Rel^{2p}_{cp} are structuring the space, five different approaches are shown and evaluated by these criteria. The final result will be that it is possible to use the same basis algorithm for Rel^{2p}_{cp} and Rel^{2p}_{cpt}.

4.1 Requirements for the Computation

The requirements for the computational model are divided in two parts. The first five criteria are more general and the last two are more dependent on the underlying model for the computation of the elementary relations. The resulting model for the computation of composite spatial relations should comply with the following requirements:

1. Applicability in 2-dimensional and 3-dimensional space.
2. The *REFO*'s extension must be taken into account.
3. Composition of the two elementary spatial relations.
4. Separate scaling for the elementary relations.

   If one elementary relation is more dominant then the other, it should be possible to use a distinct scaling for the elementary relations.[4]
5. Angular scaling.

   The angular structure of the projective relations involved should be preserved.
6. Distance concept for projective relations should be unnecessary.

   As mentioned before, the distance concept for projective relations is likely to be dropped or modified. Therefore we want to have a computational model for the compositions which does not insist on the distance.
7. The resulting degrees map into the whole applicability interval [0..1].

4.2 Computation of Rel^{2p}_{cp}

The computational model for the composite spatial relations should meet all the seven requirements. We will now present five different approaches for the computation. For each model we will point out the demands which are not complied with.

1. *Evidence Combination Calculus*

   In [Stopp et al. 94] the use of an *evidence combination calculus* (cf. [Wahlster 81]) was proposed to compute Rel^{2p}_{cp}.

[4] We are currently undertaking empirical experiments to get psychological evidence for this criterion.

$$EV_{1\wedge 2} := EV_1 + EV_2 - EV_1 \cdot EV_2, \quad EV_1, EV_2 > 0$$

The degree of applicability for a composite projective relation $DA_{Rel_{cp}^{2p}}$ was than defined as:

$$DA_{Rel_{cp}} := \begin{cases} DA_{Rel1} + DA_{Rel2} - |DA_{Rel1} \cdot DA_{Rel2}| & : \; DA_{Rel1}, DA_{Rel2} > 0 \\ 0 & : \; else \end{cases}$$

with Rel1 $\in \{left, right\}$, Rel2 $\in \{in\ front\ of, behind, above, below\}$

Figure 5 shows an example visualization of the 2-D applicability structure of "*in front of* and *left*" using a rectangle as reference object.

Valuation:
The computation of composite spatial relations using the *evidence combination calculus* needs the distance concept for the scaling. Therefore a dropping of the distance concept is not possible. There is also no *real* angular scaling and a distinct scaling of the elementary relations is difficult.

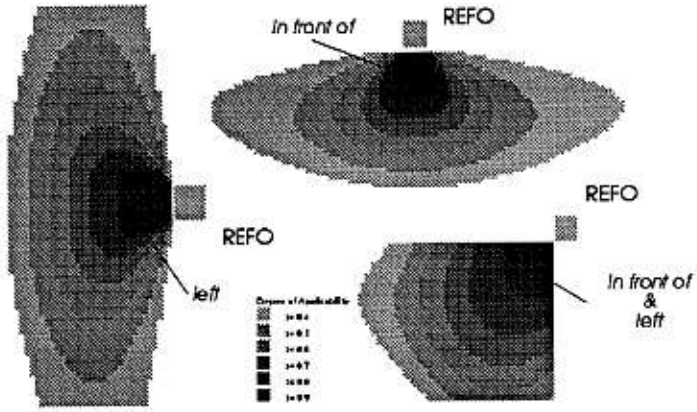


Figure 5: Applicability fields of: *left, in front of,* and their *composition* using *evidence combination calculus*

2. *Difference*

The *difference model* is the next approach in computing compositional relations we want to discuss. The formula used is:

$$DA_{Rel_{cp}} := \begin{cases} 1 - |DA_{Rel1} - DA_{Rel2}| & : \; DA_{Rel1}, DA_{Rel2} > 0 \\ 0 & : \; else \end{cases}$$

with Rel1 $\in \{left, right\}$, Rel2 $\in \{in\ front\ of, behind, above, below\}$

The *difference model* results in a high degree of applicability if the single degrees are similar and a low degree if the single degrees are different.

Valuation:
Figure 6a shows that the difference model works well in 2-D space. However it is not applicable in 3-D space as we can see in Figure 6b. This

is because the more the z-coordinate of the *LO*'s center of gravity is increasing, the more regions exist on a horizontal level with the same degree of applicability for the elementary relations. A distinct scaling of the elementary relations is also difficult in the *difference model.*

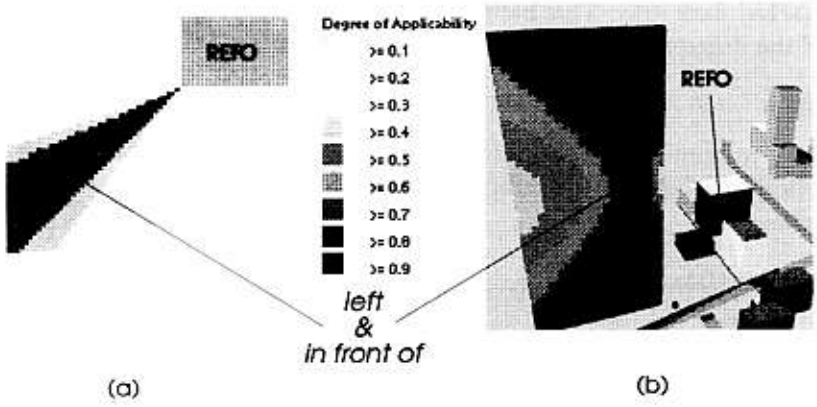


Figure 6: Example of computation for Rel_{cp}^{2p} using the *difference model*

3. *Rotation*

The *rotational model* treats the composite relation as just another elementary projective relation using a different canonical direction. The direction in 2-D space is determined by the vector reaching from the *REFO*'s center of gravity to the particular corner of the bounding rectangle, because the direction depends on the *REFO*'s shape (cf. Figure 7a).

We can now apply the algorithm for the computation of simple projective relations. The local coordinate system will be rotated to the direction of the composition and aligned to the *REFO*'s extension (cf. Figure 7b).

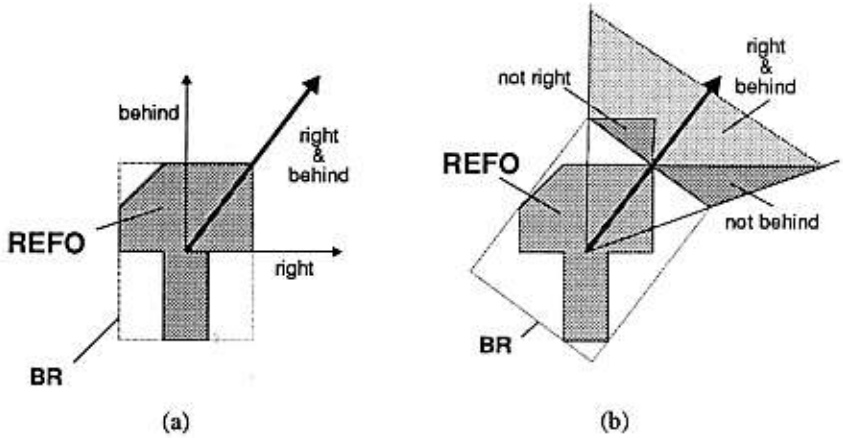


Figure 7: The *rotational approach*

Valuation:

Reducing the problem of the composition of two elementary projective relations to the computation of a projective relation with a new canonical direction seems to be elegant. However, there are regions with a high degree of applicability for the composition where only one elementary relation is applicable and the other is not. In Figure 7b these regions are marked dark gray. This happens because the result is not a combination of the degrees of applicability of the elementary relations.

4. *Minimum*

Using the minimum of the applicability of the two elementary relations results in an applicability field for the composite relation where both elementary relations are applicable.

$DA_{Rel_{cp}} := \mathrm{Min}(DA_{Rel1}, DA_{Rel2})$

with Rel1 $\in$ {*left*, *right*}, Rel2 $\in$ {*in front of*, *behind*, *above*, *below*}

Figure 8 shows, the structure of the applicability region *left and in front of* using the minimum function.

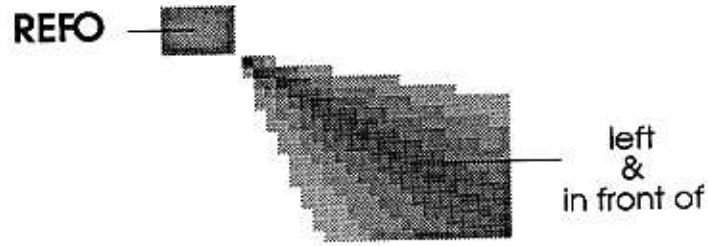


Figure 8: The *minimum approach*

Valuation:

As Figure 8 shows the resulting degrees of applicability do not map into the whole applicability interval. This is because the elementary relations have a lower degree than one if the located object occupies a position in the direction of the composite relation.

5. *Scaled Minimum*

The only problem using the minimum function is the lack of the complete range of the applicability interval. However the structure of the resulting composite relation looks exactly as it should. On a first look at the collected data in an experiment for the evaluation of spatial relations, it seems that the subjects rated the degrees of applicability on the average at 0.5 when the located object was positioned in the canonical direction of the composite relation. In this case we can easily get the complete applicability interval by multiplying the resulting degrees with a certain

scaling function S. For $x = 0.5$, S(x) must be equal to 1.0. On a first approximation we can use the constant function $S(x) = 2, \forall x \in [0, 0.5]$. However, we leave the exact definition of S open, because it could be possible that there is a higher intensification necessary for the lower degrees than for the higher. In this case we can use a linear or a spline function for S.

$$DA_{Rel_{cp}} := S(\text{Min}(DA_{Rel1}, DA_{Rel2})) \cdot \text{Min}(DA_{Rel1}, DA_{Rel2})$$

with Rel1 $\in \{left, right\}$, Rel2 $\in \{in\ front\ of, behind, above, below\}$

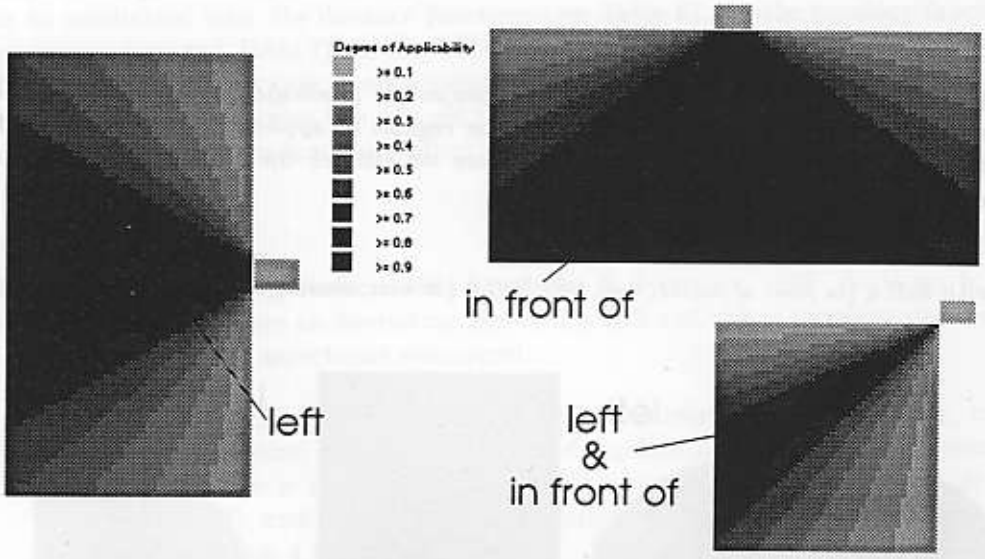


Figure 9: Applicability structure of Rel_{cp}^{2p} using the *scaled minimum approach*

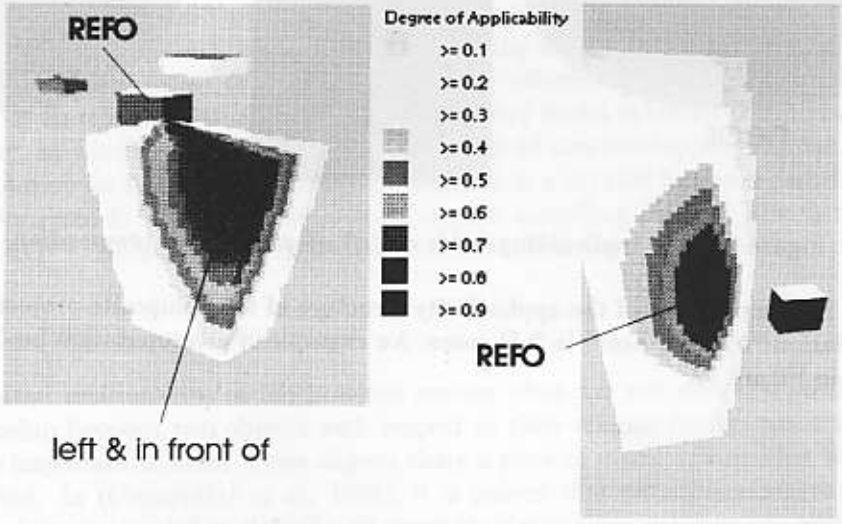


Figure 10: 3-D applicability fields of *left and in front of*

Figure 9 (2-D) and in Figure 10 (3-D) show example visualizations of Rel_{cp}^{2p} using the *scaled minimum* with $S(x) = 2$.

Valuation:

The *scaled minimum* conveys all seven requirements. It applies both in 2-dimensional and 3-dimensional space, the reference objects extension is taken into account, the angular structure is preserved and a separate scaling for the elementary relations is possible. Furthermore the distance concept for the computation of the elementary projective relations is not necessary and the resulting degrees are mapped into the whole applicability interval [0..1].

4.3 Computation of Rel^{2p}_{cpt}

A positive establishment of a composite projective topological relation also requires evidence from both elementary relations. The regions of applicability of the projective and the topological relation overlap. Therefore we can set the scaling function S to 1 when using the *scaled minimum*:

$DA_{Rel_{cpt}} := \text{Min}(DA_{Rel1}, DA_{Rel2})$

with $Rel1 \in \{in\ front\ of, behind, right, left\}, Rel2 \in \{at, near, closeto\}$

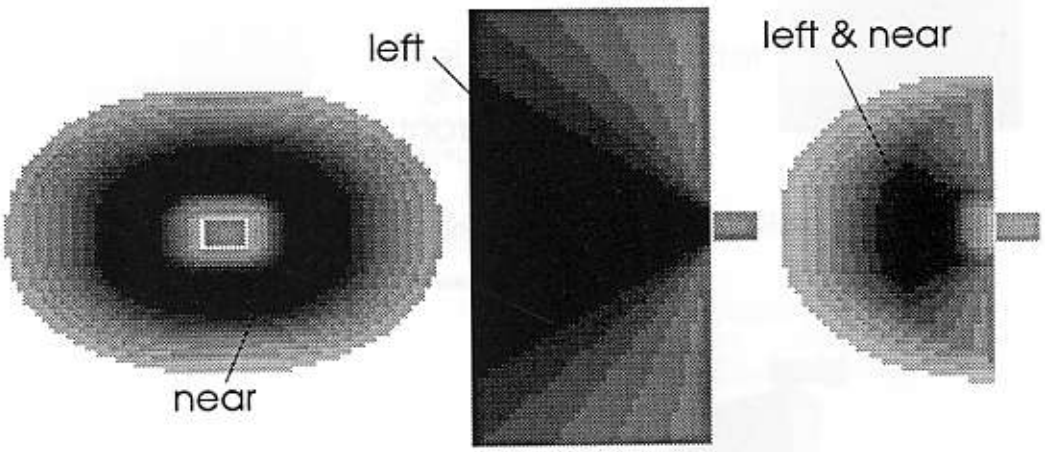


Figure 11: 2-D applicability fields of: *left*, *near*, and the *composition*

Figure 11 gives an idea of the applicability structure of the composite projective topological relation "*left* and *near*" in 2-D space. An example of a computation for 3-D space is shown in Figure 12.

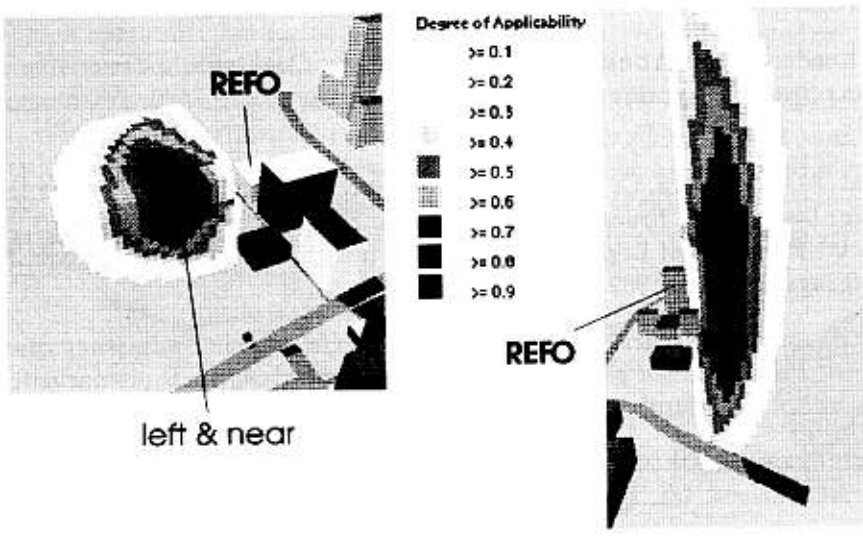


Figure 12: 3-D applicability fields of *left and near*

5. CONCLUSION

In certain situations, the set of elementary spatial relations is insufficient to localize objects exactly. We have, therefore, extended our existing computational model of elementary spatial relations to compositions which achieve for more exact localizations. The graded composite relations are defined by a combination of the elementary relations using the *scaled minimum*.

Our future research will focus on the adaptation of the exact evaluation of the computational model to the data collected in a psychological evaluation experiment and on a situation-based selection process using conceptual knowledge to determine the appropriate relation or composition for the localization of objects.

REFERENCES

[Abella & Kender 93] A. **Abella** and J. R. **Kender**. *Qualitatively Describing Objects Using Spatial Prepositions.* In: Proc. of AAAI-93, pp. 536–540, Washington, DC, 1993.

[André et al. 89] E. **André**, G. **Herzog**, and T. **Rist**. *Natural Language Access to Visual Data: Dealing with Space and Movement.* In: F. Nef and M. Borillo (eds.), Proc. of the 1st Workshop of Logical Semantics of Time, Space and Movement in Natural Language. Hermès, 1989.

[Carsten & Janson 85] I. **Carsten** und T. **Janson**. *Verfahren zur Evaluierung räumlicher Präpositionen anhand geometrischer Szenenbeschreibungen.* Diplomarbeit, FB Informatik, Univ. Hamburg, 1985.

[Egenhofer & Franzosa 91] M. J. **Egenhofer** and R. D. **Franzosa**. *Point-set topological spatial relations.* Int. J Geographical Information Systems, 5(2):161–174, 1991.

[Egenhofer & Herring 90] M. J. **Egenhofer** and J. R. **Herring**. *A Mathematical Framework for the Definition of Topological Relationships.* In: K. Brassel and H. Kishimoto (eds.), Proc. of the 4th International Symposiumon Spatial Data Handling, pp. 803–813. Zürich, 1990.

[Freeman 75] J. **Freeman**. *The Modelling of Spatial Relations.* Computer Graphics and Image Processing, 4:156–171, 1975.

[Gapp & Maaß 94] K.-P. **Gapp** and W. **Maaß**. *Spatial Layout Identification and Incremental Descriptions.* In: Proc. of the AAAI-94 Workshop on Integration of Natural Language and Vision Processing, Seattle, WA, 1994.

[Gapp 94a] K.-P. **Gapp**. *Basic Meanings of Spatial Relations: Computation and Evaluation in 3D Space.* In: Proc. of AAAI-94, Seattle, WA, 1994.

[Gapp 94b] K.-P. **Gapp**. *Einsatz von Visualisierungstechniken bei der Analyse von Realweltbildfolgen.* In: Proc. of the 1 st Workshop of Visual Computing, Darmstadt, 1994.

[Hanßmann 80] K.-J. **Hanßmann**. *Sprachliche Bildinterpretation für ein Frage-Antwort-System.* Ifi-hh-m-74/80, FB Informatik, Univ. Hamburg, 1980.

[Herskovits 86] A. **Herskovits**. *Language and Spatial Cognition. An Interdisciplinary Study of the Prepositions in English.* Cambridge, London: Cambridge University Press, 1986.

[Kochen 74] M. **Kochen**. *Representations and Algorithms for Cognitive Learning.* Artificial Intelligence, 5:199–216, 1974.

[Landau & Jackendoff 93] B. **Landau** and R. **Jackendoff**. *"What" and "where" in spatial language and spatial cognition.* Behavioral and Brain Sciences, 16:217–265, 1993.

[Logan 94] G. D. **Logan**. *Spatial Attention and the Apprehension of Spatial Relations.* Journal of Experimental Psychology, 20(5), 1994.

[Mukerjee & Joe 90] A. **Mukerjee** and G. **Joe**. *A Qualitative Model for Space.* In: Proc. of AAAI-90, pp. 721–727, Boston, MA, 1990.

[Olivier et al. 94] P. **Olivier**, T. **Maeda**, and J. **Tsuji**. *Automatic Depiction of Spatial Descriptions.* In: Proc. of AAAI-94, pp. 1405–1410, Seattle, WA, 1994.

[Peuquet 87] D. J. **Peuquet**. *An Algorithm to Determine the Directional Relationship between Arbitrarily-Shaped Polygons in the Plane.* Pattern Recognition Society, 1987.

[Rajagopalan 94] R. **Rajagopalan**. *A Model for Integrated Qualitative Spatial and Dynamic Reasoning about Physical Systems.* In: Proc. of AAAI-94, pp. 1411–1417, Seattle, WA, 1994.

[Retz-Schmidt 88] G. **Retz-Schmidt**. *Various Views on Spatial Prepositions.* AI Magazine, 9(2):95–105, 1988.

[Schirra & Stopp 93] J. R. J. **Schirra** and Eva **Stopp**. *ANTLIMA – A Listener Model with Mental Images.* In: Proc. of the 13 th IJCAI, pp. 175–180, Chambery, France, 1993.

[Stopp et al. 94] E. **Stopp**, K.-P. **Gapp**, G. **Herzog**, Th. **Laengle**, and T. C. **Lueth**. *Utilizing Spatial Relations for Natural Language Access to an Autonomous Mobile Robot.* In: L. Dreschler-Fischer and B. Nebel (eds.), KI-94. Berlin, Heidelberg: Springer, 1994.

[Talmy 83] L. **Talmy**. *How Language Structures Space.* In: H. Pick and L. Acredolo (eds.), Spatial Orientation: Theory, Research and Application, pp. 225–282. New York, London: Plenum, 1983.

[Wahlster 81] W. **Wahlster**. *Natürlichsprachliche Argumentation in Dialogsystemen.* Berlin, Heidelberg: Springer, 1981.

[Yamada et al. 88] A. **Yamada**, T. **Nishida**, and S. **Doshita**. *Figuring out Most Plausible Interpretation from Spatial Descriptions.* In: Proc. of the 12 th COLING, pp. 764–769, Budapest, Hungary, 1988.

[Zadeh 65] L. A. **Zadeh**. *Fuzzy Sets.* Information and Control, 8:338–353, 1965.

[Zadeh 68] L. A. **Zadeh**. *Fuzzy Algorithms.* Information and Control, 12:94–102, 1968.

INTEGRATED 3D MODELLING WITHIN A GIS

PETER VAN OOSTEROM, WILCO VERTEGAAL, MARCEL VAN HEKKEN
TNO Physics and Electronics Laboratory,
P.O. Box 96864, 2509 JG The Hague, The Netherlands,
Email: oosterom@fel.tno.nl.

and

TOM VIJLBRIEF
TNO Human Factors Research Institute,
P.O. Box 23, 3769 ZG Soesterberg, The Netherlands.
Email: tom@tm.tno.nl

ABSTRACT. This paper describes the architecture of a truly integrated 3D GIS. The system has two main components: an extensible database, Postgres, and an extensible geographic front-end, GEO^{++}. The database is extended with 3D geometric data types, functions, and a 3D access method. The implementation of the 3D visualization is based on the graphics standard PEX. The resulting system is again extensible and can be used as a framework for modelling and analyzing many kinds of 3D applications: geology, air-space management, terrain modelling, etc.

1. INTRODUCTION

The relational approach has proven to be very effective for developing various types of information systems. Methodologies and advanced tools are available for designing and implementing relational data models and associated applications; e.g. CASE and 4GL tools. However, the relational databases are not capable of handling spatial information, as encountered in a Geographic Information System (GIS), efficiently. We have therefore designed a 2D spatial extension (van Oosterom & Vijlbrief, 1991) and implemented this within the Postgres eRDBMS[1]. Together with a geographic front-end, called GEO^{++}(Vijlbrief & van Oosterom, 1992), this system is a complete GIS.

For certain applications the current capabilities of GEO^{++} are not sufficient; a few examples are:

1. *terrain modelling* (Douglas, 1986; Gold, 1984; Gold, Charters, & Ramsden, 1977):

[1]eRDBMS stands for extensible Relational DataBase Management System (Stonebraker & Rowe, 1986; Stonebraker, Rowe, & Hirohama, 1990). Note that the functionality in Postgres is similar to the future SQL3 standard (Robinson & Tom, 1993), as used in for example *Illustra*, the commercial successor of the Postgres research DBMS. Though we use Postgres examples in this paper, the method is general and can be applied in combination with any eRDBMS.

the GIS contains elevation data of the Earth surface (2.5D data model[2]), which can be displayed and analyzed; e.g. compute which areas are visible from a certain location;

2. *geology* (Barrera & Vázquez-Gómez, 1989; Calkins, Xia, & Guan, 1993; Carlson, 1987; Jones, 1989): the layers below the Earth surface (rock, sand, oil, water, ...) have to be modelled; e.g. for exploration by an oil company;

3. *air-space management*: the space above the Earth surface is subdivided into 3D corridors and is allocated to airplanes;

4. *fishery*: monitoring or simulating the fish population and movements (lat/long position and depth) in the Oceans;

5. *cadastre*: the cadastral boundaries in tall buildings can be true 3D; e.g. in the city of Amsterdam;

6. *cartography* (Kraak, 1988, 1992): certain cartographic techniques require 3D capabilities to visualize thematic attribute information; e.g. a PRISM map.

It is clear that we need a 3D GIS for these types of applications. There are a number of different systems available for this purpose, but they usually have one or more of the following drawbacks:

- they are limited to 2.5D data models;
- they are separate modules, hard to integrate with the GIS;
- if they are true 3D, then they are usually quite specific programs: not general purpose GIS based on a DBMS.

Therefore, we decided to design and implement our own 3D GIS-modelling environment based on the GEO^{++} system. Section 2 describes the overall architecture of this system. The subsequent Sections 3 and 4 give additional details w.r.t. the Abstract Data Types (ADTs) for 3D geometric primitives and w.r.t. the topological relationships in 3D-space. Examples of complex 3D spatial analyses are listed in Section 5. The Graphic User Interface (GUI) and visualization are the topics of Section 6. The paper is concluded in Section 7, which also contains possible future work.

2. 3D GIS ARCHITECTURE

As described in (Vijlbrief & van Oosterom, 1992), an *integrated* GIS-architecture has several advantages over a *dual* or *layered* GIS-architecture: consistency and efficiency. Both thematic and spatial data are integrated within objects (or features) and stored in the same DBMS. Further, reality as captured in the data model is translated directly

[2]In a 2.5D or Digital Terrain Model (DTM), for every location (x, y), an unique elevation z can be obtained. There are three usual ways of representing these data: contours (Watson, 1992) (in analogy with normal paper maps), TIN (Peuker, Fowler, Little, & Mark, 1976; Floriani, 1987) (Triangular Irregular Network), and grid (Mark, 1978) (with height values at regular distances; as can be found in DLMS-DTED (DMA, 1986)).

to the physical DBMS in the integrated architecture. The advantages of the integrated architecture are valid for both 2D and 3D modelling.

Weibel (Weibel, 1993) defines three levels of integration of the Digital Terrain Model (DTM)-module with the GIS:

1. *loose coupling:* GIS and DTM are separate and only exchange data sets,
2. *functional integration:* DTM-functions can be used through GIS-interface. Further, files with a common data format are used, and
3. *full database integration:* DTM and GIS are based on same DBMS and the user interface is also integrated.

Since the advantages of full database integration are also clear for true 3D-applications (not only for DTMs), we decided to follow this approach, in which both 3D data and functions are available within the same DBMS and GIS environment.

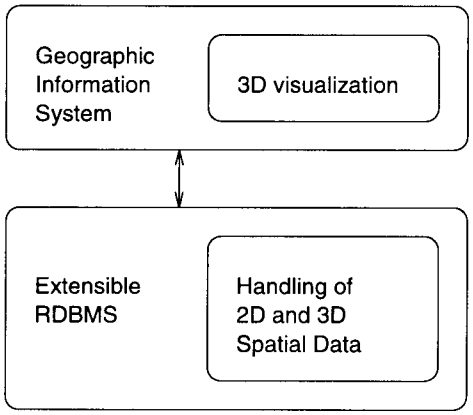


Figure 1: Architecture of an integrated 3D GIS based on an eRDBMS.

The system architecture consists of major main components: 1. an extensible RDBMS, in our case Postgres, and 2. a geographic front-end, GEO^{++}. The 3D geometric data types and functions are implemented as ADTs within the database; see the next section. Further, 3D spatial indexing is added in order to efficiently answer proximity queries. The geographic front-end is extended with 3D visualization tools. Thus we obtain an open, but integrated 3D GIS environment; see Figure 1.

3. 3D ABSTRACT DATA TYPES

The Abstract Data Types (ADTs) for the 3D Geometric Primitives are comparable to the basic 2D ADTs (van Oosterom & Vijlbrief, 1991). The 3D equivalents are: POINT3, POLYLINE3, and POLYGON3. Further, due to the nature of the 3D-space, one new type is needed: POLYHEDRON3, a true volume or body. In comparison with the 2D ADTs, there are a few additional operators; e.g. gradient and azimuth of a POLYLINE3/ POLYGON3, volume and centroid of a POLYHEDRON3 (Lee & Requicha, 1982).

Instead of 2D spatial indexing (quadtree, R-tree, etc.), 3D spatial indexing is required to enable efficient query processing. It was decided to use a 3D variant of the R-tree

(Guttman, 1984). Supporting this purpose, the type BOX3 was introduced together with an R-tree operator class for this type. The 3D data types are listed in Table 1 (Appendix A) and some examples are depicted in Figure 2.

These ADTs are sufficient for a lot of simple spatial analyses (van Oosterom & Vijlbrief, 1994). For example: "Give all regions with a surface gradient greater than 5% and that are at an elevation of 1000 meters or higher" or "Give all sand and clay soils whose surface orientation is *south*". Subsection 3.1 describes the data types in more detail and Subsection 3.2 describes the associated operators and functions.

Figure 2: Point, polyline, polygon, and polyhedron data types in 3D.

3.1 Internal/external representation

In addition to 'real' user functions, auxiliary functions are needed to read a user's input, and to show the contents of objects as output. The output should be formatted in a logical way, so that the user can understand what it represents. For example, the Postgres functions **Point3_in()** and **Point3_out()** convert, respectively, a POINT3 from and to an ASCII string. This string has the following format: three floating point numbers, separated by commas, and surrounded by brackets. For instance, the string `(1.0,2.5,-3.7)` would represent a POINT3 with an x coordinate of 1.0, a y coordinate of 2.5 and a z coordinate of -3.7.

For POLYLINE3 and POLYGON3, the formats are the same: `(nr_of_points: x1,y1,z1, x2,y2,z2, ...)`. A problem that arises with polygons is that the points that form the polygon boundary should all be in one plane. Furthermore, there should be no crossing edges. A function, named **Check3Pgn()**, has been provided to test the validity of a POLYGON3 object.

The POLYHEDRON3 data type is the most complicated of the 3D Postgres data structures. A polyhedron consists of a number of faces (polygons) that form the boundary of the object. Since a polyhedron can have any number of faces (at least 4), and all polygons can consist of any number of points larger than 2, a variable length array of variable length arrays is used to store this information. Furthermore, some representation possibilities should be considered. It may be convenient to store not only the faces, but also the points that they are built from, and/or the edges of the faces.

As for the representation problem, the decision has been made to store both the points and the faces of a polyhedron, but not the edges. This is because the edges are not really useful. They could be used to avoid redundancy (otherwise edges are implicitly defined

twice) and to check if a polyhedron is valid[3]. Therefore, it is better to compute the edges only when they are really needed. Storing the points, however, does have some benefits. In the first place, storage space is saved when the points are stored as POINT3s, and the faces consist of indexes to these, since each POINT3 consists of three `floats`, whereas an index needs only one `int`. Beyond this, it seems more logical to store a point only once and make a number of references to it, than to store this point every time it is a corner of a face. It makes the coherence between the faces clearer.

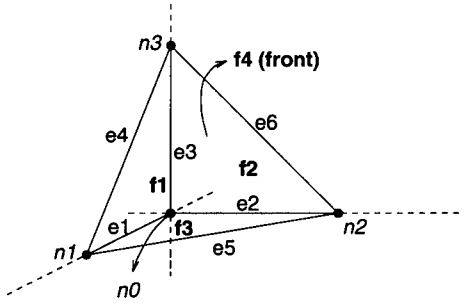


Figure 3: A simple 3D polyhedron with four faces.

The format of input and output character strings has been chosen as follows: $((n : p_1x, p_1y, p_1z, ..., p_nx, p_ny, p_nz); (m : (f\_1 : v_{1,1}, ..., v_{1,f\_1}), ..., (f\_m : v_{m,1}, ..., v_{m,f\_m}))$ where n is the number of points, m is the number of faces, and f_i is the number of point indexes in face i. Between parentheses, a list of points and a list of faces are enclosed. They are separated by a semicolon. The point list looks exactly like the POLYLINE3 and POLYGON3 format: an opening parenthesis, the number of points, a colon, all coordinates, and a closing parenthesis. The list of faces, which is also enclosed by parentheses, consists of the number of faces, a colon, and all faces, separated by commas. Each face starts with a parenthesis, followed by the number of point indexes, a colon, the indexes (separated by commas), and a closing parenthesis. For example, the polyhedron in Figure 3 would be represented by the string `((4:0,0,0, 1,0,0, 0,1,0, 0,0,1); (4:(3:0,1,2), (3:1,2,3), (3:2,3,0), (3:3,0,1)))`.

The most important reason for including a BOX3 data type is to have an economic way to access the 3D objects in a GIS using a 3D-rtree. For this purpose, the minimum and maximum x, y, and z coordinates of an object are needed. Another reason is that there are a number of functions and operators that do the same job on all or a number of the four basic 3D data types. Instead of providing these functions for all data types, they are now first converted to BOX3 and then a BOX3-function is used. Examples are minimum and maximum x, y, and z of an object. Further, intersections and unions of two objects can be approximated by using the BOX3 type.

3.2 Functions and Operators

In this subsection the functions and operators that are of interest to GIS-applications are described. Functions are the routines that actually work on the data, while operators are symbolic names given to them, that can be used within Postgres instead of

[3]A polyhedron is valid if every edge of any face is shared with exactly one other face, if no faces cross or overlap, and if all faces are valid polygons.

the full function name. For instance, a function called **Distance3PntPnt** (calculating the distance between two points) could have an operator defined as $< - >$. Instead of **Distance3PntPnt(p1, p2)** the expression **p1**$< - >$**p2** could be used, which would have the same result. These operators are also used for creating queries by means of the graphical query editor in GEO++. Further, query optimization techniques are associated with operators.

The functions and operators are subdivided into two categories: those that are explicitly suited for one data type (see Tables 2, 3, 4, and 5 in Appendix A), and those that are more general and applicable to almost any combination of two objects. The general functions can be subdivided into: the distance functions (see Table 6) and the topology functions (see next section and Table 7). In the GIS context, distances between various objects are often required. The only case where a unique distance can be obtained is with point–point operands. In all other cases one can get the minimal or maximal distance between the two objects. All distance functions return value of type `float`.

4. TOPOLOGY

Topology (Pullar & Egenhofer, 1988; Egenhofer & Franzosa, 1991; de Hoop & van Oosterom, 1992) is always an interesting aspect in a GIS and this is certainly also true in 3D. In this section two aspects are considered:

1. *classification of topology* relationships in 3D: five relationships (in, touch, cross, overlap, and disjoint) together with their boundary operators can be proven to be sufficient. This is in analogy with the 2D situation (Clementini, Felice, & van Oosterom, 1993), with only a slight modification of one definition. More details are given in Subsection 4.1.

2. *storage of topology* relationships (Molenaar, 1990; Pigot, 1992) in the data model: The POLYHEDRON3 example in Figure 4 shows two alternatives for representing the 3D body of Figure 3: either 1. store everything within the type itself (all coordinates of edges and faces) or 2. use "topological" references to faces, which in turn may contain references to edges[4]. The latter is very useful in the situation of a partitioning, in which space is subdivided into a set of non-overlapping regions. A similar solution for storing 2D topology information in a 2D GIS has been described in (van Oosterom & Vijlbrief, 1994). The complete modelling freedom is with the users of GEO++, because they can choose which model is best suited for their purpose.

4.1 Formal Topological Relationships

The spatial relationships in the previous section often are not adequate to describe the relationship between two objects with respect to their relative configuration in space. It may be important to know if two objects share a piece of space, and in what way they are connected. In (Clementini *et al.*, 1993), it is proven that only five separate topological relationships are needed to describe all possible relationships between any combination of

[4]Note that in this example the topological structure may be taken one step further (add the class nodes with a POINT3 attribute, and replace POLYLINE3 attribute in the class edges by references to nodes) or one step back (drop the class edges, and replace references to edges by POLYGON3 in the class faces).

Figure 4: Data models for 3D bodies with and without topology

```
/* without topology */
create rocks(rock_id=int4, owner=text, location=POLYHEDRON3)
append rocks(rock_id=1, owner=Shell,
   location="((4:0,0,0, 1,0,0, 0,1,0, 0,0,1);
   (4:(3:0,1,3),(3:0,2,3),(3:0,1,2),(3:1,2,3)))"::POLYHEDRON3)
    /* First the nodes, then the faces referring to these nodes */

/* with topology */
create edges(edge_id=int4, line=POLYLINE3)
create faces(face_id=int4, edge_ids=int4[])
create rocks(rock_id=int4, face_ids=int4[])
/* note that edge_ids and face_ids are both variable
   length arrays with references */

append edges(edge_id=1, line="(2:0,0,0, 1,0,0)"::POLYLINE3)
append edges(edge_id=2, line="(2:0,0,0, 0,1,0)"::POLYLINE3)
append edges(edge_id=3, line="(2:0,0,0, 0,0,1)"::POLYLINE3)
append edges(edge_id=4, line="(2:1,0,0, 0,0,1)"::POLYLINE3)
append edges(edge_id=5, line="(2:1,0,0, 0,1,0)"::POLYLINE3)
append edges(edge_id=6, line="(2:0,1,0, 0,0,1)"::POLYLINE3)

append faces(face_id=1, edge_ids={1,3,4})
append faces(face_id=2, edge_ids={2,3,6})
append faces(face_id=3, edge_ids={1,2,5})
append faces(face_id=4, edge_ids={4,5,6})

append rocks(rock_id=1, face_ids={1,2,3,4})
```

two objects from the point, polyline, and polygon types. When dropping one of the five relationships, some expressive power would be lost; adding more relationships would result in redundancy. Although the paper is about 2D objects, it almost completely covers the 3D situation, extending the relationships for use with polyhedrons without any additions and hardly any changes. For the 3D case, the five relationships are the following:

- The *touch* relationship:

$$\langle \lambda_1, touch, \lambda_2 \rangle \Leftrightarrow (\lambda_1^\circ \cap \lambda_2^\circ = \emptyset) \wedge (\lambda_1 \cap \lambda_2 \neq \emptyset) \ ;$$

  (where the interior of an object, λ°, is defined as the difference between the object λ and its boundary $\partial\lambda$.);

- The *in* relation:

$$\langle \lambda_1, in, \lambda_2 \rangle \Leftrightarrow (\lambda_1 \cap \lambda_2 = \lambda_1) \wedge (\lambda_1^\circ \cap \lambda_2^\circ \neq \emptyset) \ ;$$

- The *cross* relationship:

$$\begin{aligned} \langle \lambda_1, cross, \lambda_2 \rangle \ \Leftrightarrow \ & (dim(\lambda_1^\circ \cap \lambda_2^\circ) < max(dim(\lambda_1^\circ), dim(\lambda_2^\circ))) \wedge \\ & (\lambda_1 \cap \lambda_2 \neq \lambda_1) \wedge (\lambda_1 \cap \lambda_2 \neq \lambda_2) \ ; \end{aligned}$$

- The *overlap* relationship:

$$\langle \lambda_1, overlap, \lambda_2 \rangle \Leftrightarrow (dim(\lambda_1^\circ) = dim(\lambda_2^\circ) = dim(\lambda_1^\circ \cap \lambda_2^\circ)) \wedge (\lambda_1 \cap \lambda_2 \neq \lambda_1) \wedge (\lambda_1 \cap \lambda_2 \neq \lambda_2) \ ;$$

- The *disjoint* relation:

$$\langle \lambda_1, disjoint, \lambda_2 \rangle \Leftrightarrow \lambda_1 \cap \lambda_2 = \emptyset \ .$$

With respect to the definitions given in (Clementini *et al.*, 1993), one change was made. The original definition of the cross relation was as follows:

$$\langle \lambda_1, cross, \lambda_2 \rangle \Leftrightarrow (dim(\lambda_1^\circ \cap \lambda_2^\circ) = max(dim(\lambda_1^\circ), dim(\lambda_2^\circ)) - 1) \wedge (\lambda_1 \cap \lambda_2 \neq \lambda_1) \wedge (\lambda_1 \cap \lambda_2 \neq \lambda_2) \ .$$

This no longer holds for 3D objects. For example, the shared area of a crossing line and plane can be a point in 3D, where in 2D situations it would always be a line (assuming that the objects share any area).

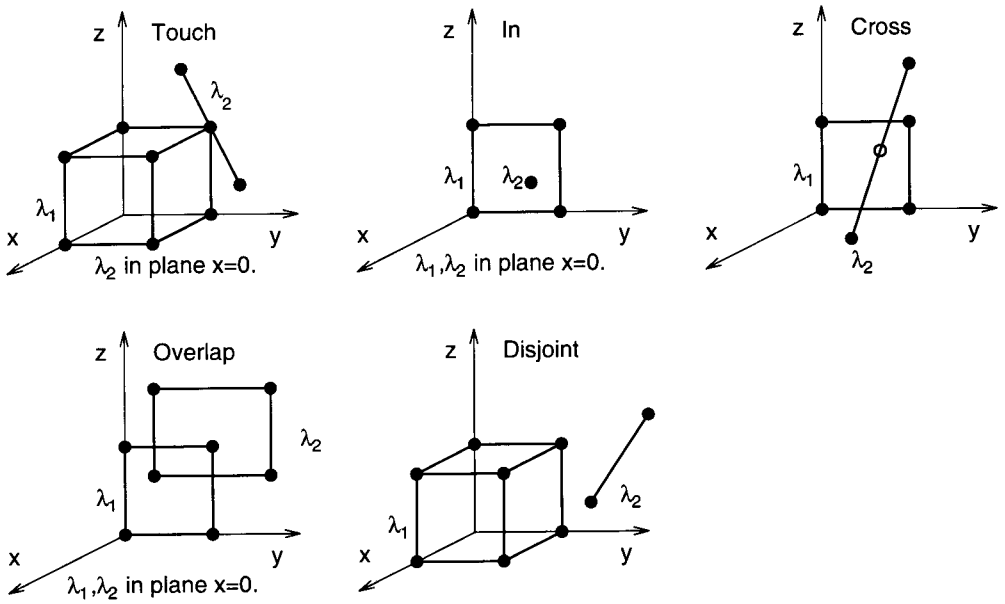


Figure 5: Examples of the five topological relationships in 3D.

With the definitions given above, a set of Boolena functions can be implemented, returning whether two objects *do* or *do not* meet the given relationship. These are given in Table 7. A sixth relation has been added, the *equal* relationship, which tests if two objects are equal. Although this could be derived from the *in* relation (two objects are equal if and only if the first object is in the second, and the second object is in the first), it is useful to create a separate function for this, which require less calculations than when using the *in* functions.

5. COMPLEX 3D SPATIAL ANALYSES

Simple analyses have already been described in Section 3. In contrast to simple analyses, complex analyses are usually done outside the DBMS. Some characteristics of complex analyses are that the computations are quite expensive and the input and output may have complex relationships. A few examples of 3D (or 2.5D) complex analyses are:

1. computation of visibility diagrams/ maps (Fisher, 1993; Floriani & Magillo, 1993; Katz, Overmars, & Sharir, 1992);
2. conversion 2.5D models, e.g. from TIN to contours (Watson, 1992);
3. 3D generalization (Ware & Jones, 1992; Jones, 1989; Floriani, 1989; Scarlatos & Pavlidis, 1991; Polis & McKeown, Jr., 1993; de Cambray, 1993), hierarchical/ multi-scale DTM's;
4. computation of watershed runoff and drainage basins;
5. finding the shortest path in a hilly terrain (Mitchel, Mount, & Papadimitriou, 1987; van Bemmelen, Quak, van Hekken, & van Oosterom, 1993);
6. compute a slice of a set of real 3D objects in order to display the internal structure; e.g. in geology.

Currently, only a few 3D analysis functions are available within GEO^{++}. However, the system is extensible and new functions can be integrated in a seamless manner within the 3D user interface of GEO^{++}; see the next section.

6. VISUALIZATION

As described in earlier papers (van Oosterom & Vijlbrief, 1991; Vijlbrief & van Oosterom, 1992), GEO^{++} is extensible. We have added several new data types before; e.g. raster and ARC2 (arc's and circles). The Postgres DBMS is extended with these ADTs implemented in C. The visualization in GEO^{++} of a new Postgres ADT is then achieved by defining a QueryShape, which is implemented in C^{++}. The core of GEO^{++} can remain unchanged, because the new type is dynamically loaded.

However, in 3D the case is quite different. The transfer from 3D-space to 2D-space, a mathematical projection, has to be defined in the "viewing pipeline" (Foley & van Dam, 1982; Singleton, 1986). The basic ET^{++} (Weinand, Gamma, & Marty, 1989) X11 toolkit, on which GEO^{++} is based, is not sufficient and has to be extended. This is done using PEX[5] (Gaskins, 1992; Rost, 1987), which is a standard 3D graphics environment on workstations; e.g. on our SparcStation under Solaris 2.3.

The 3D user interface of GEO^{++} (see Figure 6) is implemented using the extended version of ET^{++} and the BUILDER language[6]. Further, GEO^{++} itself has been modified and extended with several hooks for the 3D user interface:

[5]PEX is Phigs (ISO, 1989) 3D graphics within a X11 windows (Scheifler & Gettys, 1986).

[6]The BUILDER language (van Oosterom & Vijlbrief, 1994) can be used for efficiently developing user interfaces without "low level" programming. The BUILDER itself is also based on ET^{++}, which assures the uniform look.

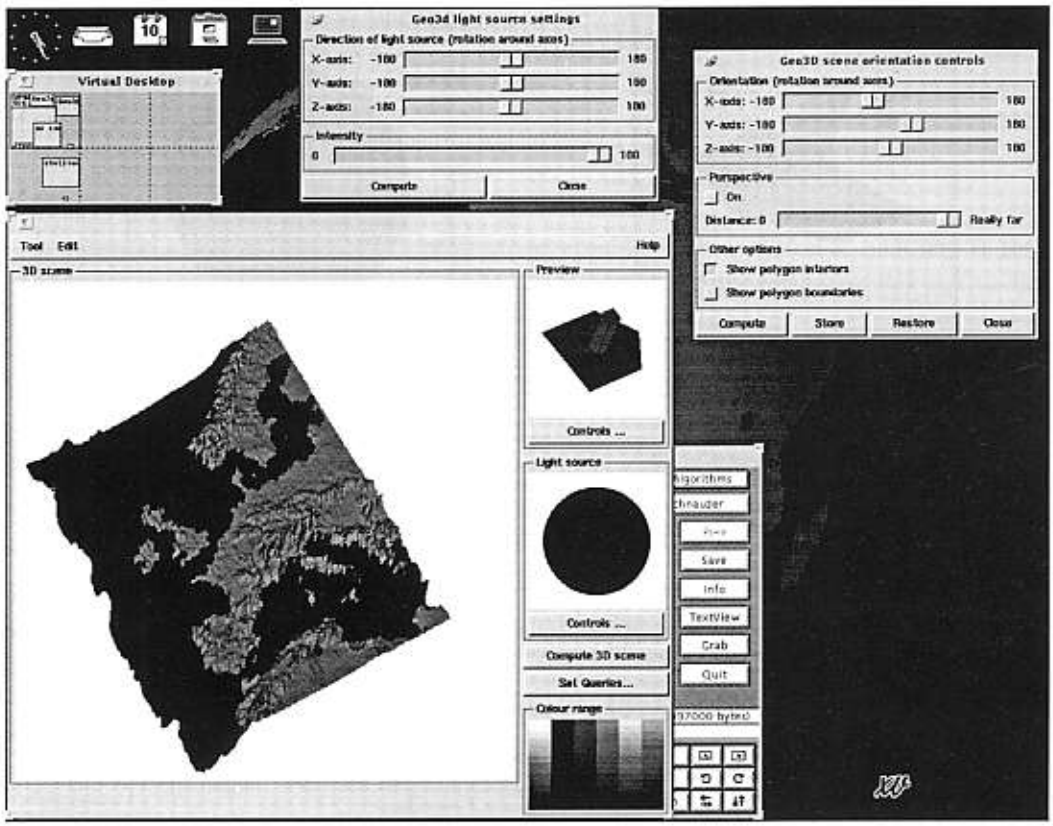


Figure 6: The 3D Geographic User Interface.

- Tuple Editor, Table Browser, and Query Composer are made available;
- Notifications of Area Changed / Query Changed in order to adjust the 3D display;
- Data model retrieval in order to obtain the specified classes with their attributes.

The 3D user interface has direct manipulation tools for setting the rotation (model orientation) and light source (shading). The *preview object* is designed so that it gives the user a good feeling of the current rotations (orientation); see Figure 7. Further, it is very simple, so it can be rotated in real time on displays without 3D graphics hardware. The user can rotate the scene by pointing at the top, bottom, left, or right part of the preview object: a *roll-and-pitch view*. Alternatively, the object can be manipulated through 3 sliders: for rotating around the x, y, and z axis.

For specifying the light source angle, a sphere shows in a consistent way from which direction the light illuminates it; see Figure 8. By using the sliders or roll-and-pitch view, the direction of the light source can be manipulated. Further, the user can select either parallel or perspective projections, and switch boundaries and interior display on/off (wire-frame or solid display); see Figure 6. More details w.r.t. the implementation of the 3D-user interface can be found in (Vertegaal, 1994).

Figure 7: The preview object.

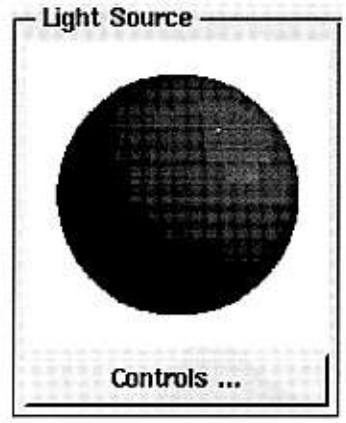


Figure 8: The reflecting sphere.

7. CONCLUSION AND FUTURE WORK

An integrated, extensible, and portable 3D GIS has been created. Note that this environment is also suitable for other 3D spatial applications, such as CAD (Computer Aided Design) (Mortenson, 1985; Mäntylä, 1988). The plans for future work will focus on GIS and include:

- coupling with virtual reality environment;
- implementation of more complex analysis functions;
- scientific visualization; e.g. of dynamic flows, or of real 3D bodies which are difficult to visualize (using transparent interiors);
- monitoring or simulating a 3D phenomenon, that is, also taking time into account;
- multi-scale DTM models supported by reactive data structures (van Oosterom, 1994);
- draping of Remotely Sensed data (or scanned paper maps) over a DTM;
- support for area patches and voxel data models.

References

Barrera, R., & Vázquez-Gómez, J. (1989). A shortest path method for hierarchical terrain models. In *Auto-Carto 9*, pp. 156–163.

Calkins, H. W., Xia, F. F., & Guan, W. (1993). GIS based 3d segmentation for water quality modeling. In *GIS/LIS Proceedings, Minneapolis, MN*, pp. 92–101.

Carlson, E. (1987). Three-dimensional conceptual modeling of subsurface structures. In *Auto-Carto 8*, pp. 336–345.

Clementini, E., Felice, P. D., & van Oosterom, P. (1993). A small set of formal topological relationships suitable for end-user interaction. In *SSD'93: The Third International Symposium on Large Spatial Databases, Singapore*, pp. 277–295 Berlin. Springer-Verlag.

de Cambray, B. (1993). Three-dimensional (3d) modelling in a geographical database. In *Auto-Carto 11*, pp. 338–347.

de Hoop, S., & van Oosterom, P. (1992). Storage and manipulation of topology in postgres. In *Proceedings EGIS'92: Third European Conference on Geographical Information Systems*, pp. 1324–1336. EGIS Foundation.

DMA (1986). Product specifications for digital terrain elevation data (DTED). Defense Mapping Agency, Aerospace Center, St Louis, Mo.

Douglas, D. H. (1986). Experiments to locate ridges and channels to create a new type of digital elevation model. *Cartographica, 23(4)*, 29–61.

Egenhofer, M. J., & Franzosa, R. D. (1991). Point-set topological spatial relations. *International Journal of Geographical Information Systems, 5(2)*, 161–174.

Fisher, P. F. (1993). Algorithm and implementation uncertainty in viewshed analyses. *International Journal of Geographical Information Systems, 7(5)*, 331–347.

Floriani, L. D., & Magillo, P. (1993). Computing visibility maps on a digital terrain model. In *COSIT'93, Elba Island, Italy*, pp. 248–269 Berlin. Springer-Verlag.

Floriani, L. D. (1987). Surface representation based on triangular grid. *The Visual Computer, 3(1)*, 27–50.

Floriani, L. D. (1989). A pyramidal data structure for triangle-based surface description. *IEEE Computer Graphics & Applications, 9(2)*, 67–78.

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

Gaskins, T. (1992). *PEXlib Programming Manual.* O'Reilly & Associates, Inc.

Gold, C. M., Charters, T. D., & Ramsden, J. (1977). Automated contour mapping using triangular element data structures and an interpolant over each irregular triangular domain. *ACM Computer Graphics, 11(2)*, 170–175.

Gold, C. M. (1984). Common-sense automated contouring, some generalizations. *Cartographica, 21*, 121–129.

Guttman, A. (1984). R-trees: a dynamic index structure for spatial searching. *ACM SIGMOD, 13*, 47–57.

ISO (1989). Information processing systems – computer graphics – programmers hierarchical interactive graphics system (phigs) – part 1: functional description. Tech. rep. ISO IEC 9592-1, International Organization for Standardization.

Jones, C. B. (1989). Data structures for three-dimensional spatial information systems in geology. *International Journal of Geographical Information Systems, 3(1)*, 15–31.

Katz, M. J., Overmars, M. H., & Sharir, M. (1992). Efficient hidden surface removal for objects with small union size. *Computational Geometry: Theory and Applications, 2*, 223–234.

Kraak, M. J. (1988). *Computer-Assisted Cartographical Three-Dimensional Imaging Techniques.* Ph.D. thesis, Delft University of Technology.

Kraak, M.-J. (1992). Tetrahedrons and animated maps in 2d and 3d space. In *Proceedings of the 5th International Symposium on Spatial Data Handling, Charleston, South Carolina*, pp. 63–71 Columbus, OH. International Geographical Union IGU.

Lee, Y. T., & Requicha, A. G. (1982). Algorithms for computing the volume and other integral properties of solids. i. known methods and open issues. *Communications of the ACM, 25(9)*, 635–641.

Mäntylä, M. (1988). *An Introduction to Solid Modeling.* Computer Science Press, Rockville, Md.

Mark, D. M. (1978). Concepts of data structures for digital terrain models. In *Proceedings of the DTM Symposium, ASPRS-ACSM, St. Louis, Missouri*, pp. 24–31.

Mitchel, J. S. B., Mount, D. M., & Papadimitriou, C. H. (1987). The discrete geodesic problem. *SIAM J. Comput., 16(4)*, 647–668.

Molenaar, M. (1990). A formal data structure for three dimensional vector maps. In *4th International Symposium on Spatial Data Handling, Zürich*, pp. 830–843 Columbus, OH. International Geographical Union IGU.

Mortenson, M. E. (1985). *Geometric Modeling.* John Wiley, New York.

Peuker, T. K., Fowler, R. J., Little, R. J., & Mark, D. M. (1976). Digital representation of three dimensional surfaces by triangulated irregular network. Tech. rep. 10, ONR Contract N00014–75–C–0886.

Pigot, S. (1992). Topological models for 3d spatial information systems. In *Proceedings of the 5th International Symposium on Spatial Data Handling, Charleston, South Carolina*, pp. 344–360 Columbus, OH. International Geographical Union IGU.

Polis, M. F., & McKeown, Jr., D. M. (1993). Issues in iterative tin generation to support large scale simulations. In *Auto-Carto 11*, pp. 267–277.

Pullar, D. V., & Egenhofer, M. J. (1988). Toward formal definitions of topological relations among spatial objects. In *Proceedings of the 3rd International Symposium on Spatial Data Handling, Sydney, Australia*, pp. 225–241 Columbus, OH. International Geographical Union IGU.

Robinson, V. B., & Tom, H. (1993). Towards sql database language extensions for geographic information systems. Tech. rep. NISTIR 5258, National Institute of Standards and Technology.

Rost, R. J. (1987). PEX PROTOCOL SPECIFICATION, VERSION 3.00. , Digital Equipment Corporation.

Scarlatos, L., & Pavlidis, T. (1991). Adaptive hierarchical triangulation. In *Auto-Carto 10*, pp. 234–246.

Scheifler, R. W., & Gettys, J. (1986). The x window system. *ACM Transactions on Graphics, 5(2)*, 79–109.

Singleton, K. (1986). An implementation of the gks-3d/phigs viewing pipeline. In Requicha, A. A. (Ed.), *Eurographics '86*, pp. 325–355.

Stonebraker, M., & Rowe, L. A. (1986). The design of postgres. *ACM SIGMOD, 15(2)*, 340–355.

Stonebraker, M., Rowe, L. A., & Hirohama, M. (1990). The implementation of postgres. *IEEE Transactions on Knowledge and Data Engineering, 2(1)*, 125–142.

van Bemmelen, J., Quak, W., van Hekken, M., & van Oosterom, P. (1993). Vector vs. raster-based algorithms for cross country movement planning. In *Auto-Carto 11*, pp. 304–317.

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. EGIS Foundation.

van Oosterom, P., & Vijlbrief, T. (1994). Integrating complex spatial analysis functions in an extensible gis. In *Proceedings of the 6th International Symposium on Spatial Data Handling, Edinburgh, Scotland.* Accepted.

van Oosterom, P. (1994). *Reactive Data Structures for Geographic Information Systems.* Oxford University Press, Oxford.

Vertegaal, W. (1994). A 3D modelling extension to Postgres/GEO++. Tech. rep. FEL-94-S164, FEL-TNO Divisie 2.

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.

Ware, J. M., & Jones, C. B. (1992). A multiresolution topographic surface database. *International Journal of Geographical Information Systems, 6(6)*, 479–496.

Watson, D. F. (1992). *Contouring: A Guide to the Analysis and Display of Spatial Data.* Pergamon Press, New York.

Weibel, R. (1993). On the integration of digital terrain and surface modeling into geographic information systems. In *Auto-Carto 11*, pp. 257–266.

Weinand, A., Gamma, E., & Marty, R. (1989). Design and Implementation of ET^{++}: A Seamless Object-Oriented Application Framework. *Structured Programming, 10(2)*, 63–87.

APPENDIX A: 3D SPATIAL ADT FUNCTIONS

This appendix contains the functions specific for one of the following 3D spatial data types: POINT3, POLYLINE3, POLYGON3, and POLYHEDRON3; see Table 1. In Tables 2, 3, 4, and 5, the in- and output functions are omitted. Table 7 contains an overview of the *topological* predicates and Table 6 lists the *distance* functions.

data type	Postgres name	components
point	POINT3	x, y, z coordinates (floats)
polyline	POLYLINE3	list of POINT3s
polygon	POLYGON3	list of POINT3s
polyhedron	POLYHEDRON3	list of POINT3s, list of faces
box	BOX3	2 POINT3s (opposing corners)

Table 1: The 3D geometric data types.

name	arguments	return value	description
Pnt2ToPnt3	POINT2	POINT3	converts point from 2D to 3D
Pnt3ToPnt2	POINT3	POINT2	converts point from 3D to 2D
Box3Pnt	POINT3	BOX3	converts POINT3 to BOX3

Table 2: Point functions.

name	argument	return value	description
Pln2ToPln3	POLYLINE2	POLYLINE3	converts polyline from 2D to 3D
Pln3ToPln2	POLYLINE3	POLYLINE2	converts polyline from 3D to 2D
Box3Pln	POLYLINE3	BOX3	converts POLYLINE3 to BOX3
Length3Pln	POLYLINE3	float	length of polyline
MinGradient3Pln	POLYLINE3	float	minimum gradient of polyline
MaxGradient3Pln	POLYLINE3	float	maximum gradient of polyline
AvgGradient3Pln	POLYLINE3	float	average gradient of polyline
MinAzimuth3Pln	POLYLINE3	float	minimum azimuth of polyline
MaxAzimuth3Pln	POLYLINE3	float	maximum azimuth of polyline
AvgAzimuth3Pln	POLYLINE3	float	average azimuth of polyline

Table 3: Polyline functions.

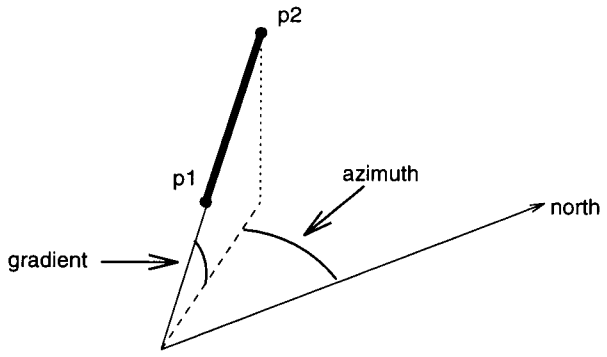


Figure 9: Definition of gradient and azimuth of a line segment.

name	arguments	return value	description
Pgn2ToPgn3	POLYGON2	POLYGON3	converts polygon from 2D to 3D
Pgn3ToPgn2	POLYGON3	POLYGON2	converts polygon from 3D to 2D
Box3Pgn	POLYGON3	BOX3	converts POLYGON3 to BOX3
Gradient3Pgn	POLYGON3	float	gradient of polygon
Azimuth3Pgn	POLYGON3	float	azimuth of polygon
GravCentre3Pgn	POLYGON3	POINT3	center of gravity
Centroid3Pgn	POLYGON3	POINT3	centroid of polygon
Area3Pgn	POLYGON3	float	surface area of polygon
Perim3Pgn	POLYGON3	float	perimeter of polygon
Check3Pgn	POLYGON3	boolean	tests if polygon is valid

Table 4: Polygon functions.

name	argument	return value	description
Box3Phn	POLYHEDRON3	BOX3	converts POLYHEDRON3 to BOX3
GravCenter3Phn	POLYHEDRON3	POINT3	center of gravity of polyhedron
Centroid3Phn	POLYHEDRON3	POINT3	centroid of polyhedron
Area3Phn	POLYHEDRON3	float	surface area of polyhedron
Content3Phn	POLYHEDRON3	float	content of polyhedron
Check3Phn	POLYHEDRON3	boolean	tests if POLYHEDRON3 object is valid

Table 5: Polyhedron functions.

name-prefix	input combinations	description
Distance3	PntPnt	distance between two points
MinDist3	all except PntPnt	minimal distance between two objects
MaxDist3	all except PntPnt	maximal distance between two objects

Table 6: Distance functions (return value is float).

prefix	object combinations	description
Equal3	PntPnt, PlnPln, PgnPgn, PhnPhn	the objects are equal
Touch3	all except PntPnt	the objects share boundary, but no interior areas
In3	all possible combinations	the first object is completely contained by the second object
Cross3	PlnPln, PlnPgn, PlnPhn, PgnPgn, PgnPhn	the objects share part(s) of their interiors, but the dimension of the shared area's boundary is lower than the higher of the dimensions of the two objects' boundaries
Overlap3	PlnPln, PgnPgn, PhnPhn	the objects share part(s) of boundaries, and the dimension of the shared area's boundary is equal to the dimension of the objects' boundaries
Disjoint3	all combinations	the objects share neither parts of boundaries, nor any area

Table 7: Topological relationships (return value is boolean).

Application Oriented Modelling

DEVELOPING A MODEL FOR GEOGRAPHIC DATA DISTRIBUTION IN A DISTRIBUTED GEOGRAPHIC INFORMATION SYSTEM

Bert Veenendaal
Department of Geographic Information Systems
School of Computing
Curtin University of Technology
GPO Box U1987, Perth, Western Australia 6001
Tel: +619 351 7676, Fax: +619 351 2819
Email: bert@cs.curtin.edu.au

ABSTRACT. As the number and complexity of geographic information systems (GIS) increases and as distributed information systems develop further, the design of GIS databases in a distributed environment becomes more important and will present new challenges for the designer. Data distribution design, the determination of what data will be placed at which site within a computer network, is a component of the overall database design process and for geographic databases must include both the spatial and nonspatial aspects of GIS data. This paper will discuss the distribution design problems for geographic data and introduce a model for distribution of data within a distributed GIS.

1. INTRODUCTION

Database design is becoming increasingly important for geographic information systems (GIS) particularly because of the volumes and complexity of geographic databases and because data and processing resources are being distributed in computer communications networks. One important component of database design, known as data distribution design, involves the determination of how the data should be allocated to storage sites in the network.

Data distribution design involves the use of data partitioning and replication strategies to translate a global schema to local schemata for the network sites at the logical design level (Figure 1). Deciding how distribution is to be done involves knowledge of both data and operations as well as any requirements or constraints which may affect it (Veenendaal and Hudson 1992). As detailed by Ceri and Pernici (1985) this information is gathered in the requirements analysis phase and used in the conceptual design to produce a global schema. This global schema can then be used in the logical design stage to analyze the distribution requirements from which partitions can be generated.

The objective of data distribution is to optimize access to the data. Ceri and Pernici (1985) identify different criteria which can be used to achieve this objective:

- maximal processing locality (data locality)
- complete locality (operation locality)
- data communication cost minimization
- cost of operations due to the distribution

The work described in this paper concentrates on using maximal processing locality as the criteria to determine how the distribution should be done. An optimal distribution solution is a very hard problem since every possible partitioning option and allocation combination would have to be considered. Therefore heuristic methodologies and techniques will be considered.

The unit of data partition is termed the "cell" (Hua et al 1993). These cells, once generated through the use of partitioning strategies, can possibly be replicated and are assigned to computer network sites. Multiple cells may be assigned to one site forming the data partition for that site. The result is that a local schema can be produced for each site.

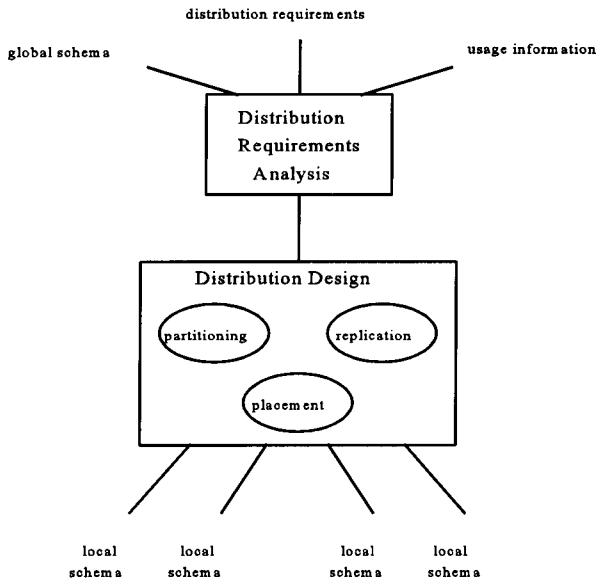


Figure 1: The process of data distribution design

So many factors influence the distribution design that the problem becomes very complicated to formulate and solve. Database information, including the data schema and relationships, and application information regarding the operation schema, the locations of the applications, and the access characteristics of the data are necessary for distribution design (Ozsu and Valduriez 1991). Further information such as storage constraints and network communication costs can also influence the distribution design but increase the complexity of the problem; their discussion is beyond the scope of this paper.

An additional complication is that all the necessary database and application information may not be available or known, and often it is difficult to predict how the applications may use the data. The emphasis in defining the distribution requirements is to gather as much information

as possible regarding the data and applications, beginning in the early stages of database design and accumulating information as the GIS is established and used.

A distributed environment is a dynamic one involving the use of different operations on various datasets at one or more sites in the network. Thus the conceptual and logical design stages of database design must involve both the relatively static (data) and dynamic (operations) requirements to produce an effective distribution design (De Antonellis and Di Leva 1985). Knowledge of the data schema and of the operation schema is important in understanding the relationship between data and operations which will affect how distribution is done. Also required is usage information which includes what data is used by what operations, the frequency and site at which the operations and data are used, and the amount of data operated upon. From this information, an initial distribution design can be determined. Of course such a design cannot be static since data and operation usage will change over time; however, dynamic data migration strategies will not be considered in this paper.

The data distribution model is being implemented in an ARC/INFO GIS workstation environment and is referred to as GEODDIST (GEOgraphic Data DISTribution). During the initial database design stage of a GIS application, the entities, operations, and their schemas are identified and represented in GEODDIST. Any available "watch" and "log" files are accessed to obtain application usage information. Various partitioning strategies are used to determine the cells which are then allocated to sites based on partitioning, polarization, and operation frequency properties.

2. REPRESENTING THE DISTRIBUTION REQUIREMENTS

For geographic data, consideration must be given to both the spatial and non-spatial data components and their interrelationships which must be modelled in the conceptual and logical design stages. The spatial data, including both geometric and topological aspects, and the related attribute data must be modelled along with the GIS operations which operate on either spatial, attribute, or both components. The difficulty lies in representing the spatial component and at the same time representing the relationships between spatial entities. Firns (1992) indicates that representing points, lines and polygons as entities within a model such as an Entity-Relationship (ER) diagram is difficult since the model becomes extremely complex and apt to incompleteness.

The representation model must distinguish between spatial entities and nonspatial entities. For spatial entities, the model must indicate if the entity is represented as points, lines or polygons. Since the spatial component can be considered as just another property of a geographic entity, it can be represented as an attribute (albeit a special one) in the ER model. For example, the ER diagram in Figure 2 indicates that *roads*, *mines* and *mapsheets* are spatial entities represented by lines, points, and polygons respectively, and *minetype* and *minerals* are non-spatial entities.

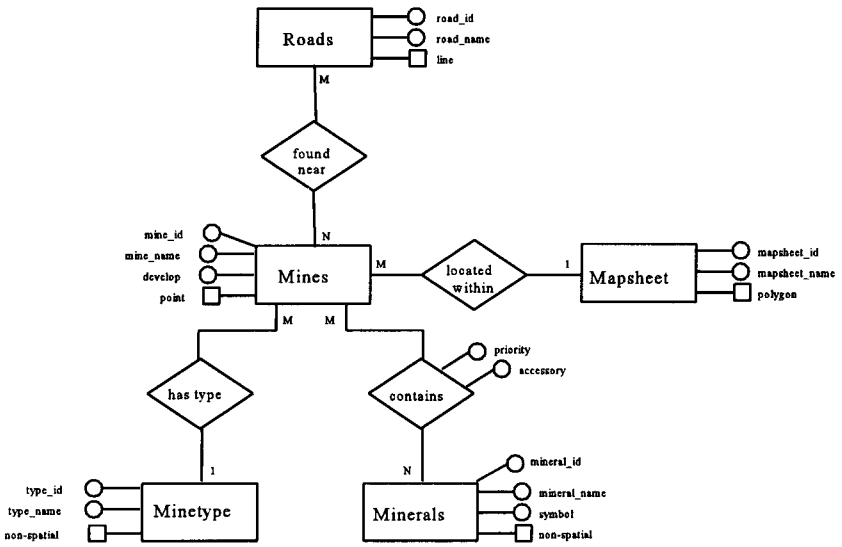


Figure 2: Entity-Relationship Diagram for Geographic Data

Representation of spatial operations is also not easy. Ceri and Pernici (1985) use an ER model for non-spatial operations which includes a means of recording the number of instances of a record, how many records were accessed, and the type of operation (keyed access, retrieval, output, write, etc.). For spatial data additional information such as the number of spatial components, as well their interrelated topology, needs to be represented for each spatial operation.

For determining the number of accesses, the spatial components cannot simply be lines or polygons since their complexity varies and hence the amount of data accessed does not directly correlate with numbers of lines and polygons. Rather the spatial primitives from which points, lines, and polygons are constructed, such as the line segment should be used as the basis for determining the number of accesses. A point is a zero-dimensional line segment and hence does directly correlate.

Thus for operations on geographic entities, both the number of attribute accesses and the number of spatial components accessed must be represented. Figure 3a illustrates an operation ER diagram for the operation which finds all mines in the *Laverton* map sheet area. The *mapsheet_id* and *mapsheet_name* attribute fields in the *mapsheet* entity are accessed sequentially (access type S) and one record (= *Laverton*) read (access type R) from a total of 98 instances of map sheets. Of the *mines* entity, 500 records of 8000 instances are selected and the *mine_name* attributes as well as the spatial locations for the selected mines are accessed for output (access type O). Note that the number of accesses and instances of spatial records is equivalent to that of attribute records since individual points directly correlate with attributes.

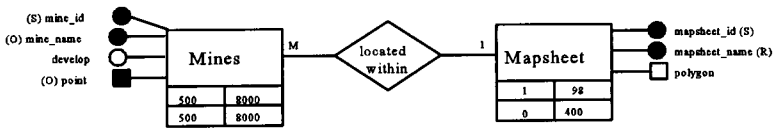


Figure 3a: Operation ER diagram for the operation: "*Identify all mines in Laverton*"

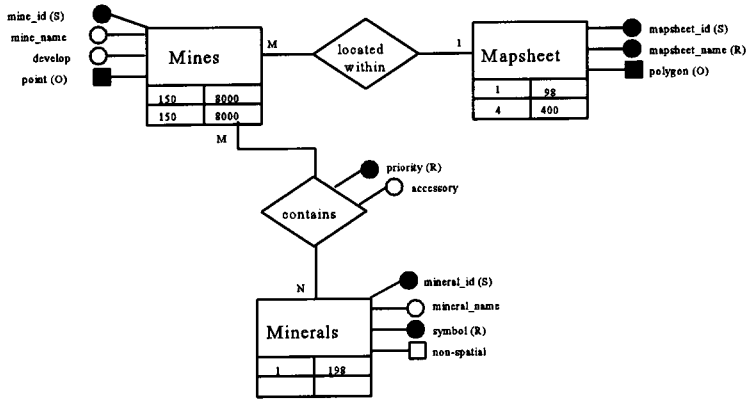


Figure 3b: Operation ER diagram for the operation: "*Display mines in Laverton with 'Au' as primary mineral*"

Figure 3b illustrates the operation ER diagram where additionally the *minerals* entity is accessed and used to constrain the display of the mine points. Four line segments of the Laverton map sheet polygon and 150 mine points are accessed for output.

The resulting operation schema as well as the data schema must be sufficiently defined, representing the information required for distribution, so that it can serve as input to the distribution requirements analysis stage. This information, particularly for the operation schema, will either come from known historical information about the applications, or it will have to be estimated. This information could obviously change in a dynamic environment.

GEODDIST uses known information about the ARC/INFO commands and usage information obtained from the "log" and "watch" files for an application. This information is obtained for each site that runs the application.

Figure 4 illustrates how GEODDIST implements the data and operation schema information. The database information regarding the entities and operations are contained in the *Entity Table* and the *Operation Table* respectively.

The relationship between the entities and operations that use them are specified in the *Operation-Entity* structure. For example, Figure 4 shows that record 1 is related to the "resel" operation which is used on the "mines" entity. Each record is related to one or more *Entity-Predicate* records which contain the information for the predicates on which the operations are executed. For example, the two predicates defined on the "mines" entity are "map_sheet_name

= 'Laverton'" and "development = 'Abandoned'", both of which are used by the "resel" operation. These predicates are used as a basis for building the partitioning and polarization tables.

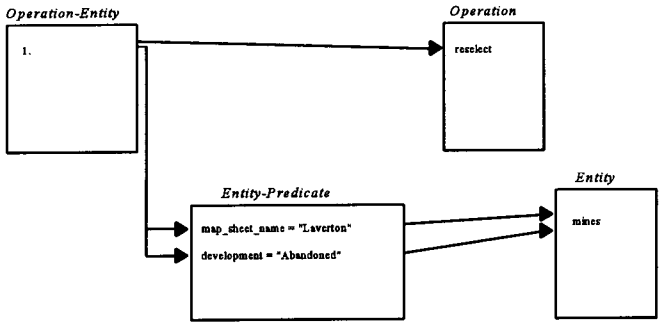


Figure 4: GEODDIST Data Structure

3. ANALYZING THE DISTRIBUTION REQUIREMENTS

In analyzing the distribution requirements, usage information will be included to generate partitioning tables, which define candidate cells among the spatial and attribute data, and polarization tables which identify how cells are polarized towards the network sites. Ceri and Pernici (1985) describe how such partitioning and polarization tables were generated for nonspatial data in their DATAID project.

3.1 Obtaining the Partitioning Tables

The candidate partitioning tables consist of selection predicates and may be derived in a similar manner as that for non-spatial data (Ceri and Pernici 1985). The difference with geographic data is that the candidate cells can be based on either the spatial data, the attribute data, or a combination of both. Table 1 gives an example of primary candidate partitionings in which the predicate is based on some spatial or nonspatial property of the entity.

Table 1: Primary Candidate Partitioning Table

	Entity	Predicate
a)	mapsheet	mapsheet_name = 'Laverton' mapsheet_name = 'Albany' mapsheet_name <> 'Laverton', 'Albany'
b)	mine	status = 'Developing' status = 'Abandoned'
c)	mine	Y-coordinate <= 32 Y-coordinate > 32

For the MAPSHEET entity the predicates are based on its attribute MAPSHEET_NAME. Note that to ensure completeness, which requires that all the records of MAPSHEET are accounted for as part of some predicate, the predicate MAPSHEET_NAME <> 'LAVERTON', 'ALBANY' is also necessary. In Table 1b the STATUS attribute is used to define predicates for a candidate partitioning for the MINES entity. Instead of an attribute being used, as in Table 1c, a spatial property, such as "location" can be used to define the predicates. Specifically, the location of mine points less than or equal to the 32 parallel (degrees latitude) is used as a predicate in this example. The converse is also used as a predicate to satisfy the completeness rule.

Candidate cells may also be derived from predicates which are obtained from other related entities. Table 2 lists some derived candidate cells for the MINES entity. The predicates in Table 2a are based on the SYMBOL attribute of the MINERALS entity which is related to MINES via the MINE_ID attribute field. In Table 2b the predicate for MINES is based on a field in the MINETYPE entity. The predicate in a derived partitioning table can also be based on a spatial relationship as is shown in Table 2c. The predicate involves the intersection (^) of mine point locations (XY-coordinates) with the 'Laverton' MAPSHEET_NAME from the MAPSHEET entity. Here the association between the two entites is based on the spatial property of location. The predicate is obtained with the use of a spatial overlay operation (eg. intersection).

Table 2: Derived Candidate Partitioning Table

	Entity	Foreign Entity	Association	Predicate
a)	mine	minerals	mine_id	symbol = 'Au' symbol <> 'Au'
b)	mine	minetype	mine_id	type = 'Underground' type = 'Bedrock' type = 'Surface'
c)	mine	mapsheet	location	XY-coord mapsheet_name = 'Laverton' XY-coord ! mapsheet_name = 'Laverton'

Associated with each predicate is the probability of the data, which satisfies that predicate, being accessed. This information can then later be used in the distribution design stage when the final cells and partitions are determined.

3.2 Obtaining the Polarization Table

Polarization indicates the relative frequencies of data being accessed by the various operations at each network site. The polarization tables determine for each operation on each predicate from the partitioning tables, the frequency at which it is accessed on each individual site in the network. The result is a 3-dimensional matrix which can then be aggregated by summing the values for each operation to produce a table showing the relative number of accesses of each predicate at each site.

Table 3 provides an example of the information required for determining the polarization of MAPSHEET by MAPSHEET_NAME for the ARCS operation. The "Laverton" mapsheet is accessed primarily (80%) at Site 1 whereas Site 3 and Site 4 do not access this mapsheet at all. The "Albany" mapsheet is accessed mostly at Site 2. The third entry in the polarization table indicates that all the mapsheets (represented as "*") are accessed 60% of the time at Site 4.

Such information is gathered for each predicate accessed by each operation at each site to form the complete polarization table. This information is used primarily for horizontal partitioning and the final site allocation in the distribution design stage.

Table 3: Polarization of MAPSHEET by MAPSHEET_NAME

Predicate	Operation *ARCS*			
	Site 1 (%)	Site 2 (%)	Site 3 (%)	Site 4 (%)
mapsheet_name = 'Laverton'	80	20	0	0
mapsheet_name = 'Albany'	10	60	30	0
mapsheet_name = *	10	30	0	60

Often the operations on the data are associated more with individual users (who may move around the network) or even types of tasks that a user may perform, rather than be associated directly with the network sites themselves. The idea of defining a task has been used in performance evaluation to identify a common series of spatial operations from which workloads can be estimated (Goodchild and Rizzo 1986). These tasks can then be identified with one or more sites so that the frequencies of access for each site can be obtained.

3.3 Obtaining the Operation Frequency Table

In addition to the partitioning and polarization data, information regarding the frequency of occurrence of each operation is required. Table 4 gives an example of frequencies for some GIS operations at four sites.

Table 4: Frequency of occurrence of some operations on four sites

Operation	Frequency of Occurrence			
	Site 1 (%)	Site 2 (%)	Site 3 (%)	Site 4 (%)
arcs	25	25	30	20
polygonshades	30	30	20	20
intersection	50	20	10	20
mapextents	25	30	20	25

The next step is the distribution design phase which uses the distribution requirements and information to determine how the partitioning should be done, to which site the partition should be allocated, and what replication, if any, should take place.

4. GEOGRAPHIC DATA PARTITIONING

Partitioning of non-spatial data involves horizontal and vertical partitioning of records (Ceri and Pernici 1985, Ozsu and Valduriez 1991). Geographic data involves spatial partitioning for the spatial data in addition to horizontal and vertical partitioning for the attribute data (Veenendaal and Hudson 1992, Veenendaal 1994).

4.1 Spatial and Horizontal Partitioning

Because the attribute and spatial data are very closely linked in a GIS, the spatial partitioning of points, lines, and polygons will also involve a horizontal partitioning (Veenendaal and Hudson 1992). The horizontal partitioning is done on the associated attribute data where instances of the attributes are linked to the corresponding spatial data in the resulting partitions.

Spatial and horizontal partitioning should thus be done concurrently. The objective is to increase the processing locality and thereby decrease the amount of data to be transferred between sites. The predicates that will actually form the cells must be determined from the primary and derived candidate partitioning tables.

All predicates which are accessed below a minimum access rate based on the number of data accesses and the selectivity rate would not be suitable as cells or partitions since the cost of partitioning and re-joining when required by the processing operations would be greater than the processing cost if left unpartitioned.

The selectivity rate indicates the number of accesses of a predicate relative to the total number of accesses to the entity that this predicate is defined on. The selectivity rate is calculated as follows.

$$PS_i = \#\text{ of accesses of } P_i \,/\, \text{total } \#\text{ of accesses of } E_j$$

where P_i = predicate i
PS_i = predicate selectivity rate for P_i
E_j = entity j where P_i is defined on E_j

The total number of accesses to Ej include all accesses to other predicates that are defined on Ej. A predicate selectivity rate of greater than 50% would indicate that the records defined by the predicate are accessed more frequently than for all the other predicates (defined on the same entity) combined indicating that this predicate is a good potential candidate for partitioning.

The number of data accesses for a predicate can be calculated as follows.

PC_i = # of line segments accessed for P_i * frequency of access for P_i

where PC_i = predicate access count for P_i

Note that, as previously explained, the line segment is used as the basis for determining the number of accesses for points, lines, and polygons. The predicate access rate can then be calculated as follows.

$PR_i = PS_i * PC_i$

where PR_i = predicate access rate for P_i

As an example, predicates with a selectivity of 50% and with an access count of 80 would have an access rate of 40. If a minimum access rate of 60 is used, then this predicate should not be used for spatial/horizontal partitioning. A more complex calculation of the predicate access cost could include the cost of operations for accessing and processing the predicates if they were individually partitioned. The minimum predicate access value would be based on the number of accesses where the cost of partitioning and maintaining these records defined by the predicate is equivalent to the cost of processing these records if left unpartitioned.

The remaining predicates are then identified as the spatial/horizontal cells. To ensure that the partitioning for each entity is complete, the converse of each cell must be included. So if the predicate "MINE: STATUS = 'DEVELOPING'" is chosen as a spatial/horizontal cell, then the predicate "MINE: STATUS <> 'DEVELOPING'" must be used as the converse cell. All the MINE data can then be found in one or more cells. Care must be taken that the minimum term predicate (Ozsu and Valduriez 1991) is used so that multiple predicates defined on the same entity do not involve overlapping data.

4.2 Vertical Partitioning

Vertical partitioning involves the splitting of attribute tables where some of the domains may be placed in a new cell. As with horizontal partitioning, vertical partitioning should only take place to improve the locality of processing and reduce the number of data transfers between sites. An attribute selectivity rate can be calculated in a manner similar to the predicate selectivity rate for horizontal partitioning.

AS_i = # of accesses of A_i / total # of accesses of E_j (A_i is a domain in E_j)

where A_i = attribute domain i
AS_i = attribute selectivity rate for A_i

An attribute selectivity rate of greater than 50% indicates that this attribute domain for E_j is accessed more than any other accesses to E_j and would be a good candidate for partitioning.

A more complex calculation for determining vertical partitioning would include the relationships between attribute domains. Ozsu and Valduriez (1991) describe the use of an attribute affinity matrix and clustering techniques to group related attributes for partitions.

With vertical partitioning, the new partitions of attribute data must still be linked to the associated spatial data. This means either that any partitions, placed on a site other than where the spatial data resides, must be remotely linked to the spatial data, or the spatial data must be replicated and placed on each site where the attribute data partitions reside. Because a GIS requires a strong link between the attribute data and the spatial data and because the cost of replicating an entire spatial dataset is high, vertical partitioning is often not a desirable option for partitioning, especially where the spatial component is voluminous or complex. Horizontal partitioning, because of the close link to spatial partitioning, is a much preferred option.

Mixed partitioning involves applying both horizontal and vertical partitioning iteratively (Ceri and Pernici 1985, Ozsu and Valduriez 1991). The order in which they are done affects the partitioning and is important in determining an optimal partitioning. For example a data set could first be horizontally partitioned. One or more of the resulting smaller fragments could then be further partitioned using vertical partitioning. However because of the complexity involved, finding an optimal solution to decide which partitioning strategy to use and the order of each is very hard.

5. ALLOCATION OF CELLS

Once the cells are determined, the allocation of these to partitions on computer network sites can be determined. The allocation is based on the number of relative accesses for each operation on each cell at each site. The relative access values are calculated as follows.

$$R_{ijk} = PS_i * OF_{jk} * PA_{ij} * PO_{ijk}$$

where R_{ijk} = relative access value for operation j on predicate i at site k
PS_i = predicate selectivity rate for predicate i
OF_{jk} = frequency of operation j on site k
PA_{ij} = number of record accesses of operation j on predicate i
PO_{ijk} = polarization of predicate i for operation j on site k

The relative access value R_{ijk} is calculated for each predicate (which forms a cell) with each operation on each site resulting in a 3-dimensional matrix. The effective locality access values for each cell on each site can then be calculated.

$$L_{ik} = \sum_{j=1}^{n} R_{ijk}$$

where L_{ik} = locality access value for predicate i on site k

Each cell can then be assigned to the site with the highest locality access value.

$$A_i = \max_k \{ L_{ik} \}$$

where A_i = site allocation for predicate (cell) i

Each site will consist of one or more cells which form the partition of data for that site. From these partitions and their allocations, the local schema can be constructed for each site.

Replication of cells among the sites can occur particularly in instances where the locality access values are high for two or more sites. However the costs of updating and maintaining replicate copies must also be considered before any replication is done.

The decision regarding what data should be replicated is often done in a separate step after the partitioning process is completed as this simplifies the problem (Ceri and Pernici 1985). However, particularly with spatial data, some partitioning may result in some of the data also being replicated. It is difficult therefore to separate the partitioning processes totally from that of replication.

Some replication is already inherent in the partitioning strategies particularly with vertical and spatial partitioning. In vertical partitioning when the domains are separated into separate cells, the key field must be duplicated in the new records being formed for each cell. Veenendaal and Hudson (1992) explain that with spatial partitioning for lines and polygons, the line endpoints or polygon boundaries will be replicated as lines and polygons are separated. If the lines or polygons are themselves split by partitioning, then not only are new lines and polygons created with replicated endpoints or boundaries, but the associated attributes are also replicated. The cost of maintaining and merging these partitions is relatively high since the operations required for spatial merging and the reconstruction of topology are processing intensive.

6. CONCLUSIONS

The data distribution design problem is a complex one which becomes more difficult when applied to geographic data and its spatial component. There are many variables involving the spatial and attribute data and the GIS applications that influence the distribution. This makes the development of the GEODDIST geographic data distribution model a very difficult and complex task as indicated in this paper.

Distribution design is not simply a matter of dividing up the data and allocating them to individual computer network sites. Rather, the relationships inherent in the data and its use must be considered, namely: spatial, inter-attribute, spatial-attribute and topological relationships. The data, operations, and the inter-relationships must be adequately represented and database and application information gathered for the distribution analysis.

The partitioning, polarization, and operation frequency tables are constructed from information regarding the structure of the data, usage of data by operations, frequency of usage at various sites, etc. Obtaining such information can be difficult and predicting the usage of data and operations can be even more difficult, particularly in a dynamic environment. It is, however, important to collect as much information as possible and begin this collection early in the database and application design process. Such knowledge can then be built up as the applications are implemented and the data used.

REFERENCES

Ceri, S. and B. Pernici (1985) DATAID-D: Methodology for Distributed Database Design, in Computer-Aided Database Design: The DATAID Project. A.Albano, V. De Antonelis, and A. Di Leva (Editors), Elsevier Science Publishers B.V., North Holland.

De Antonellis, V. and A. Di Leva (1985) DATAID-1: A Database Design Methodology. Information Systems, 10:2, pp 181-195.

Firns, P.G. (1992) On the Notion of a Spatial Entity in the Context of Data Modelling for Spatial Information Systems. Proceedings of the Fourth Colloquium of the Spatial Information Research Centre, University of Otago, Dunedin, New Zealand, May.

Goodchild, M.F., and B.R. Rizzo (1986) Performance Evaluation and Workload Estimation for Geographic Information Systems. Proceedings of the Second International Symposium on Spatial Data Handling, Seattle, Washington, U.S.A., July.

Hua, K.A., C. Lee, and H.C. Young (1993) Data Partitioning for Multicomputer Database Systems: a Cell-based Approach. Information Systems, 18:5, pp. 329-342.

Ozsu M.T. and P. Valduriez (1991) Principles of Distributed Database Systems. Prentice-Hall International, Inc., Englewood Cliffs, New Jersey.

Veenendaal, B. and D. Hudson (1992) Distributing GIS Data in a Computer Network. Proceedings of GIS'92 Symposium, Vancouver, British Columbia, Canada, February.

Veenendaal, B. (1994) Partitioning Vector Data in a Distributed Geographic Information System. Technical Report 5, Curtin University of Technology, Perth, Western Australia.

MODELLING AND STORAGE OF ROAD NETWORK DATA

VOLKER WALTER and DIETER FRITSCH
Institute for Photogrammetry
Stuttgart University
Keplerstr. 11
D-70174 Stuttgart, Germany
Email: Volker.Walter@ifp.uni-stuttgart.de

ABSTRACT. This paper is an introduction to some aspects of modelling and storage of road network data. After describing the mapping of topological data onto a relational database the GDF data model is presented. An overall look at the conceptual data model and a more detailed view at the attribute concept shows the potential of GDF. An example is given of the representation of the topology at different GDF levels.

The second part of the paper presents possible applications of GDF data. The merging of GDF data with national basis information systems is discussed. Especially the fusion of GDF with the German digital topographic information system called ATKIS is investigated. Arising problems are described and a look into future work is given.

1. INTRODUCTION

The standardized data exchange format in Europe of road traffic data is the Geographic Data File (GDF). GDF data are already completely acquired in Germany and the Netherlands and will be captured for all of Europe in the future.

At the same time many national institutions are beginning to acquire spatial data for national basis spatial information systems. This means that many topographic features will be captured more than once if there is no way of data exchange between GDF and other data models. The GDF data model is very powerful and its data can be used for a lot of applications which are not concerned of the traffic environment.

GDF data can be mapped easily onto a relational data model. A typical task of a system which uses GDF data is the search for shortest paths in graphs which represent a road network. Therefore a method is required to store the graph in a way that the topology can be accessed very efficiently. The storage of geometric data in ringlists provides a fast access to the topology.

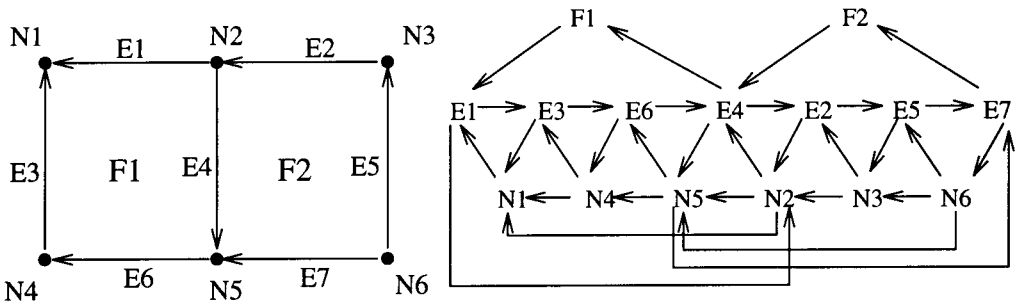


Figure 1. Ringlists representing a graph

2. MAPPING OF TOPOLOGICAL DATA ONTO A RELATIONAL MODEL

Figure 1 shows a graph which represent a part of a road network. There are six nodes which are connected with seven edges. The nodes represent intersections and the edges represent road elements. Also there are two faces (the term for areas in GDF) which are formed by the edges.

It is not difficult to map the topology onto a relational model. Faces are represented by their surrounding edges and edges by a beginning node and an ending node. For every node a pair of coordinates must be stored. Therefore three relational tables are needed to store all topological relations.

Typically the relational model requires a lot of join operations. But the join operation is the problematic factor in pathfinding because the tables get very large. Even by using indices at the relational tables many harddisk accesses must be done. Therefore a method is required to store the graph in spatial clusters by using spatial indices.

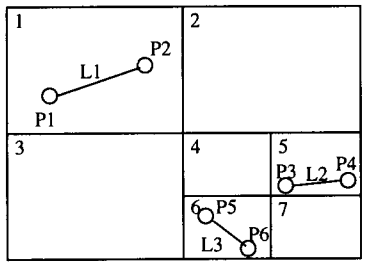
Figure 1 shows the representation of the graph by using ringlists. Topological elements (nodes, edges and faces) are divided into master and detail elements. The master element face consists of the detail elements edges. Edges are master element and detail element at the same time. They consist of the detail elements nodes. The master element points to its first detail element. The detail element points to the next detail element until the last detail element is reached which points back to the master element.

The ringlists are realized with pointers that direct points to the adress on which the following element is stored. By using this memory structure one can navigate very efficiently through a graph. To select the nodes which can be reached from a specific node one only has to follow the corresponding pointers without computing any join.

The storage of the ringlists can be done with the method of "quad-tree" (Sammet 1986). First a rectangular working area is defined. This working area is assigned to a cell where the graphic elements (points, lines and areas) will be stored. The relational database saves this cell as a BLOB (Binary Large OBject).

The cell has a fixed maximum size. When the storage capacity of a cell is exceeded, four new cells are created which each cover a quarter of the original cell. The graphic elements are distributed onto the new cells. With these cells a spatial index is created. This allows the user to load only a local part of the whole database and speeds up the data processing. The cells will be administrated in a relational table, the cell table

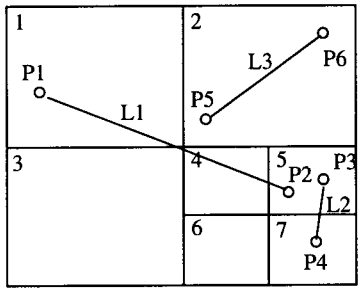
(figure 2). If all elements have to be loaded which lie in a cell one only have to load the corresponding BLOB from the database.



ID	Elements
1	P1 L1 P2
2	
3	
4	
5	P3 L2 P4
6	P5 L3 P6
7	

Figure 2. The cell table

Because the working area is divided into cells there also exist elements that can be found in several cells. Thus if an element is stored into several cells one has to find all the cells involved. Two tables are used for this procedure. The element table stores the cells where the elements can be found. The cell is computed by using the coordinate of the element. This coordinate is either determined directly from the parameters of the element (e.g. at points) or derived from the element structure (e.g. at lines from the starting point). In contrast to the cell table, the element table does not contain the element data but only the ID. The element table creates an index to the cell table which allows access to the elements without having to search the cell table in advance. Because of this index the access time does not depend on the data volume.



ID	Elem
1	
2	
3	L1
4	L1
5	L1
6	
7	L2

Reference table

ID	Elem
1	P1
1	L1
2	P5
2	P6
2	L3
3	
4	
5	P2
5	P3
5	L2
6	
7	P4

Element table

Figure 3. The element table and reference table

Elements which lie in multiple cells also receive entries in the reference table in addition to the entry in the element table. The reference table lists all the cells of

an element that are in contact with this element. Figure 3 shows an example for the element table and the reference table.

This form of storage of geometric data is applied in the geographic information system SICAD/open (SICAD 93). SICAD/open is a trademark of the geographic information system offered by Siemens Nixdorf, Munich, Germany.

3. GEOGRAPHIC DATA FILE

GDF is a standard for the exchange of digital data for vehicle navigation systems in Europe. The first draft of GDF was developed in an EUREKA project (EUREKA is an initiative of the european industry to promote pre-competitive industrial collaboration). GDF has been revised several times and the present standard is GDF 2.1. The GDF documentation (Heres et.al. 1991) provides a detailed description in eight parts. The documentation defines features, attributes and relations. Features are real world objects such as roads, buildings or traffic signs. The documentation also contains information about the representation of the features, quality requirements, global data and about physical storage.

3.1 The conceptual data model

Figure 3 gives an overview of the conceptual data model in GDF. The diagram has been constructed according to the conventions of NIAM modelling (Nijsen Information Analysis Method, Nijsen and Halpin 1989). The centre of the diagram is formed by the feature. Features can be point features, area features or complex features which can consist of any type of feature. The subtype relationship between the different feature types is represented with an arrow. It can be seen that point features, line features and area features are subtypes of simple features which are again subtypes of general features.

Every feature belongs to *not more than one* feature class. This is represented by an arrow above the rectancle which represents the relation between feature and feature class. The point at the circle which represents the feature means that the feature belongs to *at least one* feature class. The combination of not more than one and at least one results in *exactly one*. Therefore every feature class can contain any number of features and every feature belongs to exactly one feature class. This is a 1:n relationship between feature and feature class.

Feature classes are defined recursively and so it is possible to build any form of class hierarchy. Every feature class belongs to exactly one feature theme which is not represented in this diagram. The following feature themes are defined in the GDF catalogue: Roads and Ferries, Administrative Areas, Settlements, Land Use Units, Brunnels, Railways, Waterways, Road Furniture and Services.

Feature can take part in a semantic relationship with any number of other features and can have any number of attributes. Typical attributes for roads are "Road Width" or "Maximum Height Allowed".

Point, line and area features are represented by nodes, edges and faces. Between these elements exist the topological relationships described above. For isolated nodes the appropiate face must be stored additionally.

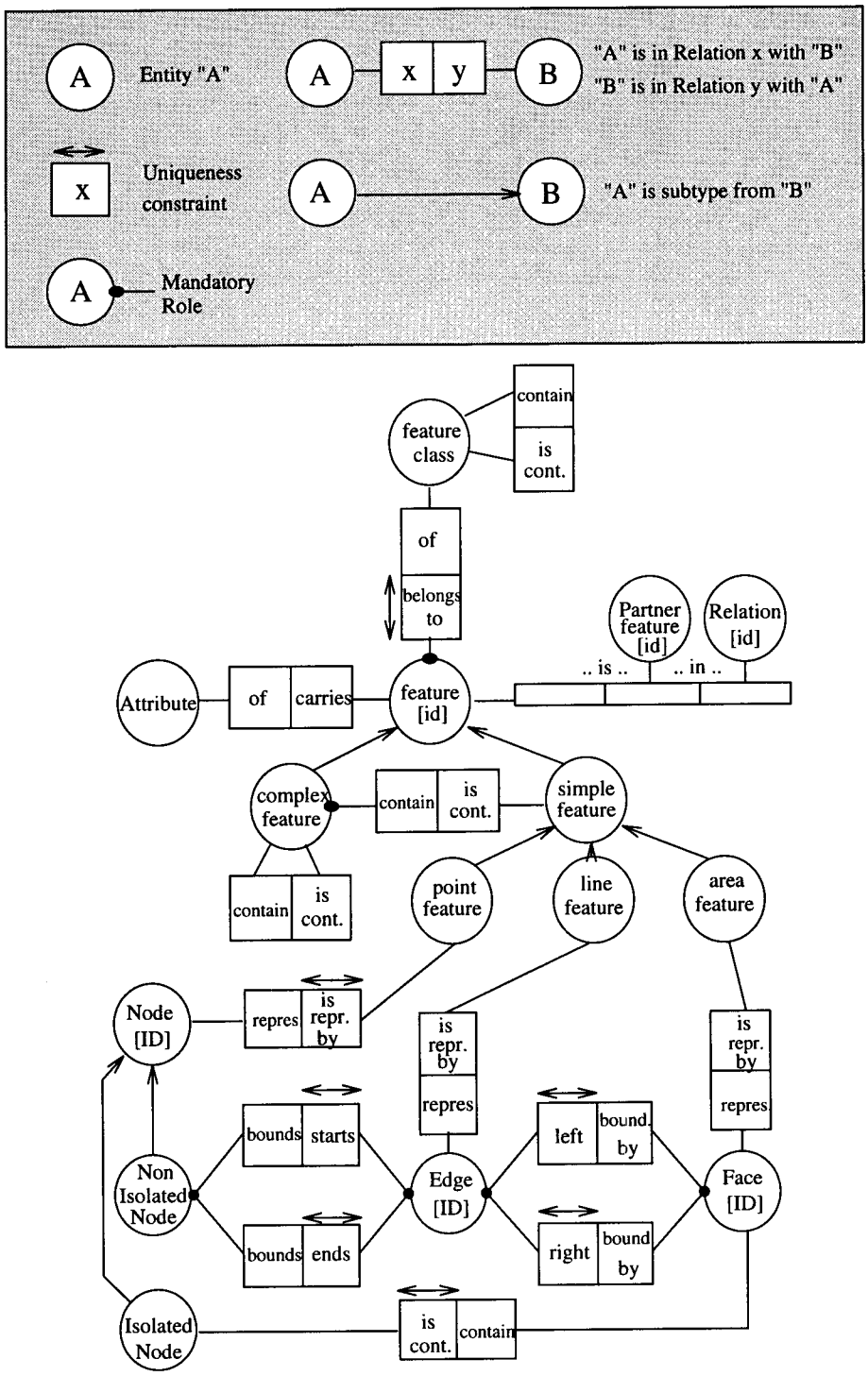


Figure 3. The GDF conceptual data model

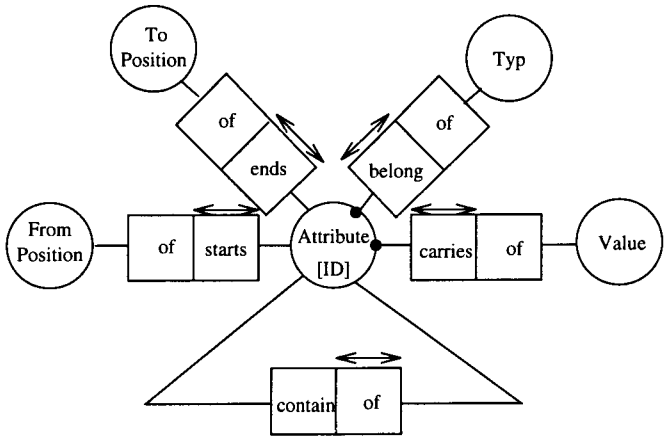


Figure 4. The GDF attribute concept

3.2 The GDF attribute concept

Attributes can be related to every feature and every semantic relationship in GDF. GDF has a powerful attribute concept which allows the use of segmented, complex and time dependent attributes. Figure 4 shows the data model for GDF attributes.

Segmented attributes can only be used for line features. Segmentation means the reference of a certain part which the attribute is valid for. A narrow part of a street can be represented by the segmented attribute "Road Width" that divides the street in several parts that the attributes are general valid for. This form of modelling is suitable for the modelling of temporary changes in the road network, such as roadworks. It is not neccesary to create a new feature to model a roadwork because it is sufficient to define a segmented attribute. Therefore temporary changes can be inserted and deleted in the database without a lot of changes in the structure.

Complex attributes consist of subattributes which again can be themselve complex. Therefore it is possible to define any attribute hierarchy. An example for a complex attribute in GDF is the attribute "Traffic Sign Information" which consists of the subattributes "Traffic Sign Class", "Direction", "Symbol on Traffic Sign", "Textual Content of a Trafffic Sign" and "Value on Traffic Sign" .

Time dependent attributes are the most frequent complex attributes. Every attribute can be used in combination with the attribute "Validity Period" (e.g. no parking from 9:00 am to 12:00 am).

3.3 Representation of the road network

The road network can be represented at different levels of aggregation. Figure 5 shows an example with the different levels. The "Original" shows the topographic features which should be acquired, a road network and a border between two different administrative areas.

The level 0 representation represents all (simple) features by a planar graph. In figure 5 b) nodes are represented with circles and intermediate points with an rectangle filled with a cross. It can be seen that a node is inserted at the intersection between

the street and the border. This makes it possible to do exact topological queries to the database.

Every feature class of level 0 will be represented in an own independent layer of the layer 1 representation. Figure 5 c) shows the layer of the road elements. For example traffic guiding is implemented in this level with an onboard display.

The level 2 representation represents only complex features and their relations between them (figure 5 d)). Complex features are for example "Intersections" which consists of the simple features "Road Elements" and "Junctions". This allows a short time optimal route finding on a generalized niveau. For optimal route finding it is sufficient to store only the complex features without geometrical representation in the database.

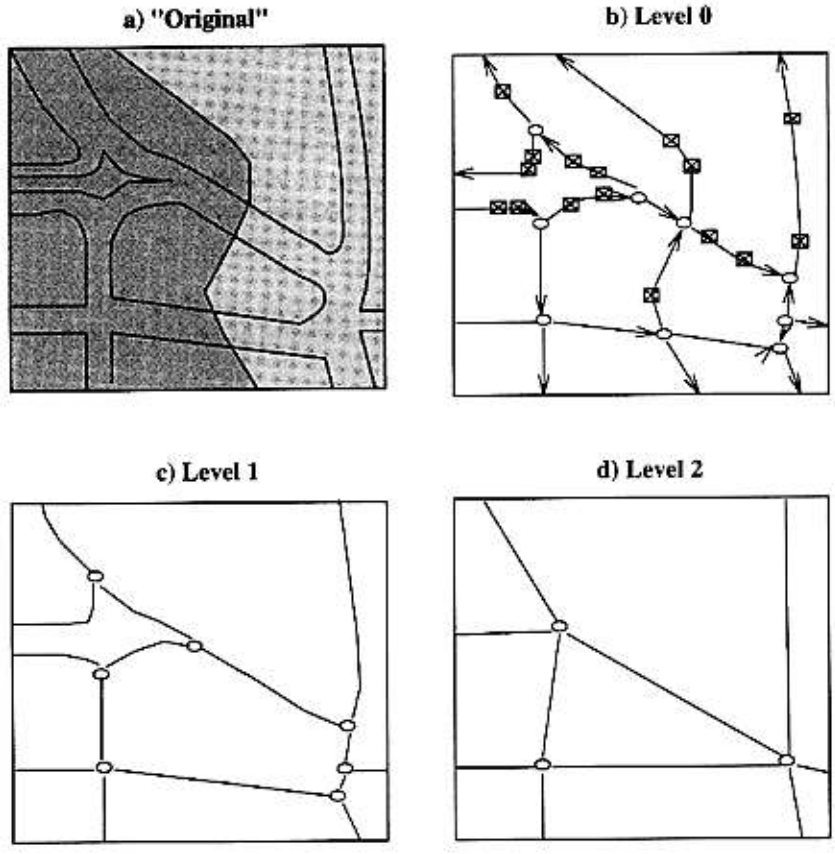


Figure 5. Different levels in GDF

4. APPLICATIONS OF GDF DATA

The GDF standard was originally designed for applications in the area of road environment, for example vehicle navigation, traffic management systems or intelligent vehicle highway systems (IVHS). But the feature catalogue also defines a number of features which are not directly concerned to the road network. They range from administrative areas (e.g. county, district or community) over landuse (e.g. woodland, lake or industrial area) to services (e.g. theatre, pharmacy or ski-lift). Because of this variety of features (and their belonging attributes) there are many other applications of GDF data possible.

Geographic informations systems can use GDF data as a basic data stock. The GDF data model is flexible and can be easily enlarged for specific applications. Because of the high density of the road network in Europe GDF data provide an excellent spatial reference which will be updated regularly and provide the users with well defined up-to-date data. The data content specification catalogue defines quality requirements, completeness requirements and up-to-dateness of the acquired data.

Therefore GDF data could play the same role as national basis information systems. But if there is no way of data exchange between national basis information systems and GDF the data must be acquired twice. Because data capture and update is very expensive and time consuming redundand data should be absolutely avoided. This means that every way of data exchnage between national basis information systems and GDF has to be found.

5. MERGING OF GDF AND NATIONAL BASIS INFORMATION SYSTEMS

Because of the increasing demand of digital spatial data, national basis information systems will be built up in most of the European states. Data exchange between these information systems and GDF would offer many advantages:

- The completion of the systems could be accelerated.
- Features and attributes which are not acquired in one system but in the other could be imported in the system without additional aquisition time and therefore would expand the spectrum of potential users of the system.
- Updating data can cause a lot of problems and would only have to be done in one system.
- GDF is already a European standard. In the future, national basis information systems should be linked together to a whole European database. This is only possible if the concepts of the different databases are similar. GDF could be a reference for the design of national basis information systems.

6. MERGING OF ATKIS AND GDF

6.1 The ATKIS concept

In October 1986, it was decided on establishing the digital topographic and cartographic information system, called ATKIS, in Germany (ADV 1988). ATKIS partitions the landscape into topographic features. The features belong to seven feature classes which are control points, settlements, traffic, vegetation, waters, areas and relief. In the first step (until the end of 1995) 60 different feature types shall be captured for Germany in the scale of 1:25,000 (a precision of +/- 3 meter when dealing with traffic ways otherwise +/- 10 meter) and afterwards updated regularly.

ATKIS data are to be stored and retrieved by means of the Digital Landscape Model (DLM). Its visualization has to be realized according to the Digital Cartographic Model (DKM). Besides the scale 1:25000 - the most detailed landscape representation called DLM 25 - there are further levels of data aggregation in the scales of 1:200,000 and 1:1,000,000. GDF and ATKIS correspond the most at the level 1:25000. Therefore, bidirectional links in this paper refer to this level.

6.2 Problems

Because ATKIS and GDF do not base upon the same data model there will emerge problems when transfering data from one data model to the other. One of the main problem is the different feature definition in the two models. This means, that a feature of the landscape is captured in one model as one feature and in the other model as several features. The reason originate from the different applications. In GDF a street is captured much more detailed than in ATKIS.

Figure 6 illustrates this problem. In GDF a physical separation between two parts of a road leads to a representation by two different "Road Elements". In ATKIS roads are represented through the middle axis independent of physical separation. The number of lanes is represented by an attribute in ATKIS. Every change in an attribute value leads to different object parts in ATKIS. The more complex the intersection the more problems will arise. A typical complex intersection is a highway crossing. It seems to be easier to convert GDF data into ATKIS data than vice versa because of the more detailed representation of GDF data.

A further problem is that ATKIS does not allow for relations between features. ATKIS supplies the user only with a basic stock of features along with their attributes. Relations are very suitable for the mapping of information like "Prohibited Turn" or "Right of Way". It is not possible to store all information of GDF data in ATKIS.

The attribute concept of GDF is much more powerful than the one in ATKIS. ATKIS accepts only features with simple attributes whose values can possess a maximum length of seven bytes where as GDF also allows complex atributes with any length.

Provided that it is possible to convert one data format into the other a further problem will arise. It is necessary, for example for updating, to match the data sets. Because of systematic and random errors introduced through digitizing the matching of the data is not a trivial problem. Figure 7 shows two data sets of an intersection which are overlayed.

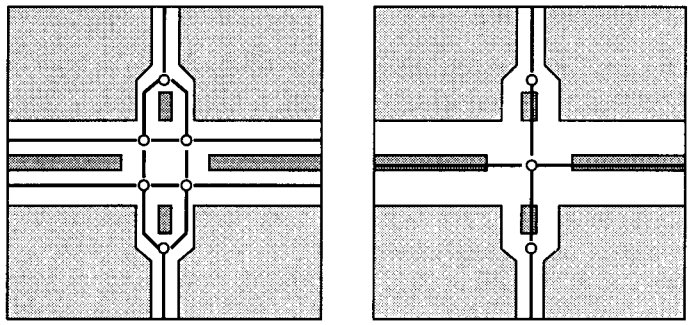


Figure 6. Different feature definitions in GDF (a) and ATKIS (b)

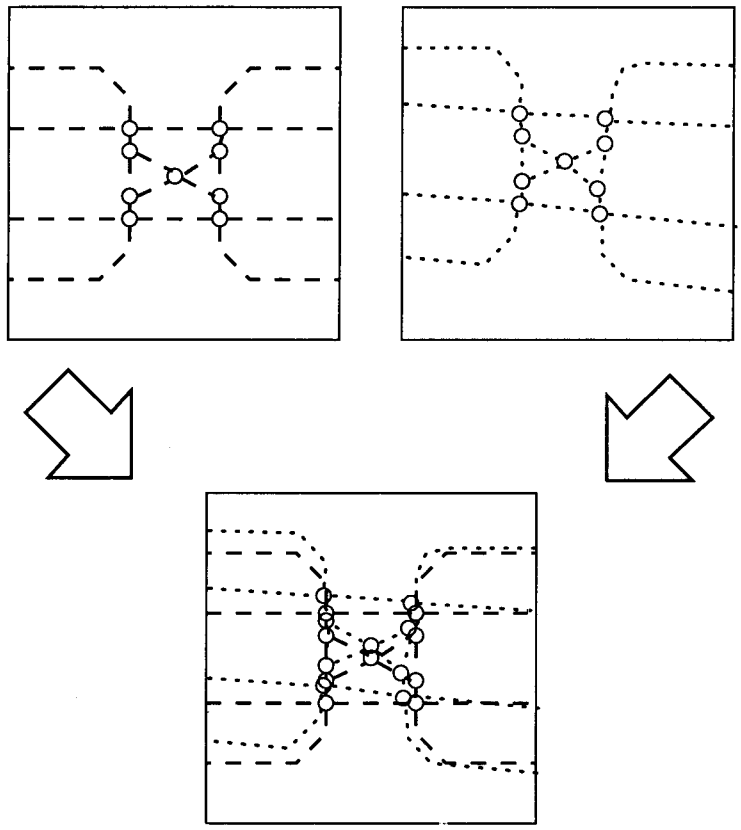


Figure 7. Systematic and random errors introduced through digitizing arising when matching the two data sets

6.3 Outlook

As a first step of merging the two data models the GDF data model will be mapped on the simplier ATKIS model. This will be done with the help of the relational GIS product SICAD which supports the use of ATKIS features. The topology will be stored as described above. Attributes and semantic relationships will be stored in relational tables. A first investigation can be found in Walter and Fritsch (1994).

In a next step GDF data will be generated from ATKIS data and vice versa. These data will be compared with original captured data. This allows to give some quality statements about the generated data. The generated data will be matched with the original data to examine the updating of GDF data with ATKIS data and vice versa.

The first acquisition step (DLM 25/1) of ATKIS is supposed to be available not before 1996 for the whole area of Germany. The second aquisition step (DLM 25/2, 168 features and 111 attributes) would offer the possibility to extend the ATKIS data model. Requirements of designing the ATKIS data model will be formulated as a result of this study.

7. SUMMARY

This paper introduces problems of modelling and storing road network data. It was shown that the storage of the topology of a road network is very efficient by using ringlists. The mapping of ringlists onto a relational model was presented. This method of mapping provides a spatial index on the data.

The Geographic Data File (GDF) is a standard for the exchange of digital road network data. The conceptual data model of GDF was presented. The attribute concept which allows the use of segmented, complex and time dependent attributes was examined. By using complex objects the topology can be represented at several levels of generalisation. The different levels were presented with examples.

GDF data can be used by a lot of applications which must not directly concerned to road environment. Especially the data exchange between national basis information systems and GDF was discussed. With an example of the German digital topographic and cartographic system (ATKIS) the problems of merging GDF data with a national basis information system were described. Finally a look into future work was given.

ACKNOWLEDGEMENTS

This research work is carried out under doctorial fellowship sponsored by Siemens Nixdorf Information Systems, Munich, Germany, which is gratefully acknowledged.

REFERENCES

ADV (1988) *Amtlich Kartographisches-Topographisches Informationssystem (ATKIS)*, Arbeitsgemeinschaft der Länder der Vermessungsverwaltungen der Bundesrepublik Deutschland, Bonn.

Heres et. al. (1991) *GDF-Documentation Vol. 1 - Vol 8.*, Task Force EDRM.

Nijsen G. M. and Halpin T.A. (1989) *Conceptual Scheme and Relational Database Design - A fact oriented approach*, Prentice Hall, Sydney.

Samet H. (1986) *Hierarchical Spatial Data Structures*, Design and Implementation of Large Spatial Databases, Proceedings of the first Symposium SSD 1989, Santa Barbara, Californien.

SICAD (1993) *SICAD-GDB-X V1.0*, Siemens Nixdorf Informationssysteme AG, AP internationales Druckzentrum, Munich, Germany.

Walter V. and Fritsch D. (1994) *GIS Data Structures for Vehicle Navigation Systems*, Proceedings of the 6th Canadian conference on GIS, Ottawa, Vol. 1, pp 489 - 501.