Bulletin of the Technical Committee on

Data Engineering

September, 1993 Vol. 16 No. 3 **IEEE Computer Society**

Letters

Letter from the Editor-in-Chief . . . . . *David Lomet* 1

Special Issue on Geographical Information Systems

Letter from the Special Issue Editor . . . . . *Kyu-Young Whang* 3
The Design of GODOT: An Object-Oriented Geographic Information System . . . . . *Oliver Günther and Wolf-Fritz Riekert* 4
Efficient Spatial Query Processing in Geographic Database Systems . . . . . *Hans-Peter Kriegel, Thomas Brinkhoff, and Ralf Schneider* 10
Spatial Indexing: Past and Future . . . . . *Hongjun Lu and Beng-Chin Ooi* 16
Interactive Spatial Directories . . . . . *Brian C. Schmult, H. V. Jagadish, and S. Kicha Ganapathy* 22
High Performance R-trees . . . . . *Christos Faloutsos and Ibrahim Kamel* 28
Using the Holey Brick Tree for Spatial Data in General Purpose DBMSs . . . . . *Georgios Evangelidis and Betty Salzberg* 34
Definition of Line-Line Relations for Geographic Databases . . . . . *Max J. Egenhofer* 40
ESPRIT Project EP 6881 AMUSING . . . . . *Paolo Giulio Franciosa and Maurizio Talamo* 46

Conference and Journal Notices

1994 International Conference on Applications of Databases 51
1994 International Conference on Data Engineering 52

Letter from the Editor-in-Chief

I am happy to report to you that the Data Engineering Bulletin is now available via electronic distribution. Below are the instructions on how to deal with the electronic mail server that does the distribution.

```
Send e-mail to the TC on Data Engineering server at:

                    tcdata@crl.dec.com

Commands should be sent in the subject line.  Only one command per message
is permitted.  The commands are:
     enroll              Use this to receive an application to become a member
                         of the TC.
     application         Use this when returning the application.  You will
                         be registered as a member of the TC.
     send <filename>     Use this to receive an issue of the Bulletin.
                         Only one file per "send" request is allowed.
     index               Use this to receive complete instructions and a list
                         of the currently available issues.  The index gives
                         you a filename for each issue.

For example:
                    send march93-letfinal

results in your being sent by e-mail the March, 1993 issue of the Bulletin
formatted for letter size paper, and including all postscript figures.
```

I would like to thank Win Treese of Digital's Cambridge Research Staff for making the modifications to our report server that enable us to provide electronic distribution of the Bulletin.

As of my writing, 345 people have enrolled as TC members. All members will be notified via e-mail whenever an issue of the Bulletin is published. They will be able to request the issue by following the above instructions. Indeed, re-enrolled members should already have received a notification that electronic distribution has commenced and that the September issue is now available, as well as the March and June issues.

The hardcopy version of the Bulletin will only be available to those that pay for a subscription. I wish it were possible to provide specifics on hard copy distribution. Unfortunately, we have still not resolved this with the Computer Society. The Computer Society currently provides printing and distribution services and these do not present problems. The snag is in exactly how the fees will be collected and at what frequency.

The current issue of the Bulletin is on Geographic Information Systems and was edited by Kyu-Young Whang. I think you will agree that it presents an interesting cross-section of the work going on in Geographic Information Systems. Kyu-Young is the last editor who was appointed by Won Kim and he will now retire as an editor of the Bulletin. I thank him for agreeing to continue to serve as an editor during this transition phase and for his handling of this issue.

The December issue of the Bulletin is on query processing and is being handled by Goetz Graefe. This issue captures the current state of the commercial art in query processing and will include papers from a good cross-section of the database vendors. Its subject is in line with my desire to have the Bulletin be a vehicle for propagating knowledge of current industrial practice.

I am gratified by the progress that has been made so far and am confident that the Bulletin will be established in a way that permits it to prosper over the long term. Electronic distribution goes a long way toward that goal. When complemented with subscription based hardcopy distribution, we should finally have what we need.

David Lomet
Editor-in-Chief

Letter from the Special Issue Editor

Geographic Information Systems (GISs) have gained popularity recently thanks to their application to diverse areas such as land management, facility management, environment management, and cartography. An increasing number of organizations, including Government agencies and private companies, have been adopting GISs for efficient management of large volumes of data. The objectives of this issue are to raise the GIS technical issues in the database research community, to summarize the current state, and to project future research directions in this exciting area with active practical use.

The GIS storage system has a major effect on performance. Many commercial GISs use as their underlying storage systems proprietary file systems or, more recently, relational database management systems. Due to complexity in the spatial and non-spatial data structures required of GIS applications and the special characteristics of their queries, conventional techniques have not seemed adequate. Recent research efforts seek to enhance the modelling power and performance of storage structures and query processing algorithms for GISs. The papers here appear in an order that presents the systems issues first, then detailed algorithms, and finally an introduction to a multi-national ESPRIT project.

The paper by Günther and Riekert describes the design of GODOT, an object-oriented GIS. It has four-layers consisting of a commercial object-oriented database system, an extensible GIS kernel, a collection of base components such as the query processor, and several user interface modules.

Kriegel, Brinkhoff and Schneider propose an architecture of query processing in the GIS consisting of four steps: 1) scaling down of the search space, 2) geometric filtering, 3) efficient transfer of exact geometry, and 4) efficient processing of the exact geometry. Their paper also proposes specific techniques for each step, which are based on the notions of five-corners, scene organization, and TR*-trees.

The paper by Lu and Ooi surveys spatial indexes, shows their evolution, and classifies them according to the techniques for handling nonzero-sized (i.e., non-point) objects. It subsequently discusses future research directions on spatial indexing.

Schmult, Jagadish, and Ganapathy present the architecture of the Interactive Spatial Directory (ISD) system being built at At&T Bell Laboratories and discuss research issues on modelling, management of heterogeneity, spatial indexing, and system performance.

The Faloutsos and Kamel paper describes two R-tree variants: the Hilbert R-tree for good clustering of R-tree nodes in a centralized system and the MX R-tree for good performance on a parallel system. It also proposes a proximity measure for distance between rectangles.

Evangelidis and Salzberg present a spatial index structure, the hB^{Π}-Tree for which a general method for concurrency and recovery is applicable. Such a tree can be embedded in a general purpose database system, extending its functionality to spatial indexing. It can also be used in a GIS.

Egenhofer identifies 33 topological relations between line objects for which geometric interpretations can be given. This algebraic topology approach compares interiors, boundaries, and exteriors of the lines. The relations identified can be applied to spatial queries relating lines in GISs.

Finally, the paper by Franciosa and Talamo introduces a GIS effort within the ESPRIT project called AMUSING. The objective of AMUSING is to define the features and principles of a next generation GIS architecture. The project involves nine organizations in six European countries.

I would like to thank the authors for their excellent contributions and their cooperation. I particularly appreciate the efforts of Franciosa and Talamo, who had to submit their paper on very short notice. Hans Schek deserves thanks for finding this ESPRIT GIS effort. Finally, I would like to acknowledge the help from Ju-Won Song at KAIST, who collected and formatted the papers.

Kyu-Young Whang
Korea Advanced Institute of Science and Technology(KAIST)
email: kywhang@cs.kaist.ac.kr, kywhang@eclipse.stanford.edu

The Design of GODOT: An Object-Oriented Geographic Information System*

Oliver Günther
Institut für Wirtschaftsinformatik
Humboldt-Universität zu Berlin
Spandauer Str. 1
10178 Berlin, Germany
guenther@faw.uni-ulm.de

Wolf-Fritz Riekert
FAW Ulm
Postfach 2060
89010 Ulm, Germany
riekert@faw.uni-ulm.de

Abstract

This paper describes the design of an object-oriented geographic information system called GODOT (Geographic Data Management With Object-Oriented Techniques). GODOT has a four-layer architecture, consisting of (i) a commercial object-oriented database system; (ii) an extensible kernel with classes and methods for representing complex spatial and non-spatial data objects; (iii) a collection of base components for query processing, graphics, database administration, and data exchange; and (iv) several user interface modules. The conceptual basis for the system is a data model with three categories of objects: Thematic objects, geometric objects, and graphic objects. The GODOT approach facilitates the management of complex geographic and environmental information and leads to an extensible system architecture.

1 Introduction

Geographic and environmental information systems operate on very complex spatial and non-spatial data structures. Relational databases are of only limited use for modeling this complexity. For that reason, most commercial geographic information systems (GIS) rely on specialized file systems or other proprietary solutions for storing the data.

The idea behind GODOT is to develop a GIS prototype on top of a commercial object-oriented database system (OODB) by adding appropriate classes and methods. With this approach, GODOT differs from most other recent GIS developments:

- GODOT's object-oriented data model allows the representation of highly *complex* geographic information (e.g., in environmental applications) as a network of geographic objects.
- The GODOT data model is *extensible* by user-defined classes and methods.
- GODOT's underlying OODB is a general purpose system which allows for both spatial data and ordinary tabular data to be seamlessly *integrated* within the same environment.

*The GODOT project is funded by the Environmental Ministry and the Information and Communication Agency of the State of Baden-Württemberg, Siemens Nixdorf Informationssysteme AG, and Siemens AG.

- GODOT is based on a commercial OODB and therefore participates in *new developments* on the database market. This may concern features such as query language standards, graphical tools, transaction management, and distributed processing.

Earlier related work was mostly based on OODB research prototypes, such as PROBE [3] or O_2 [6]. Several other projects were based on extensible database systems, such as DASDBS [7], or POSTGRES [5]. A more recent effort to use POSTGRES for the management of large amounts of geographical and environmental data is the SEQUOIA 2000 project [8].

Section 2 describes the four-layer architecture of the GODOT system. In Section 3 we present the GODOT data model with its three categories of objects: thematic objects, geometric objects, and graphic objects. Section 4 gives a brief overview of the current state of the implementation.

2 System Architecture

The GODOT system has a four-layer architecture (Fig. 1), consisting of:

1. A commercial OODB
2. An extensible kernel with classes and methods for the representation and management of complex spatial and non-spatial data objects
3. A collection of base components for query processing, graphics, database administration, and data exchange
4. Several user interface modules, including a C/C++ program interface, a UNIX command interface and a graphical user interface based on the X Window System and OSF/Motif

In the sequel we discuss those four layers in turn.

2.1 OODB and Kernel

The GODOT kernel is implemented directly on top of the commercial OODB ObjectStore [2]. ObjectStore is a fully object-oriented database system in the sense of the OODB Manifesto [1]. This covers in particular the basic database functionalities, including transaction management and indexing. Some of these functionalities have been adapted and extended for a more efficient management of spatial data.

The GODOT kernel contains the definition of classes and methods that are crucial for the representation and management of geographic information. In particular, the implementation of the GODOT data model (see Section 3) is located here. Furthermore, the spatial clustering and indexing components will be located in the kernel. We are currently in the process of evaluating a number of index structures for this purpose. As all the other GODOT components, the kernel is implemented in the object-oriented programming language C++.

The GODOT data model can be extended by additional classes and methods. This facilitates the customization of the system to serve any particular application.

2.2 Base Components

The *query component* contains several GIS-specific language elements to enhance the query language of ObjectStore. It interacts directly with the interface modules described in Section 2.3. ObjectStore allows the call of a user-defined method within a query; this feature has been used to extend the query language with spatial and topological predicates.

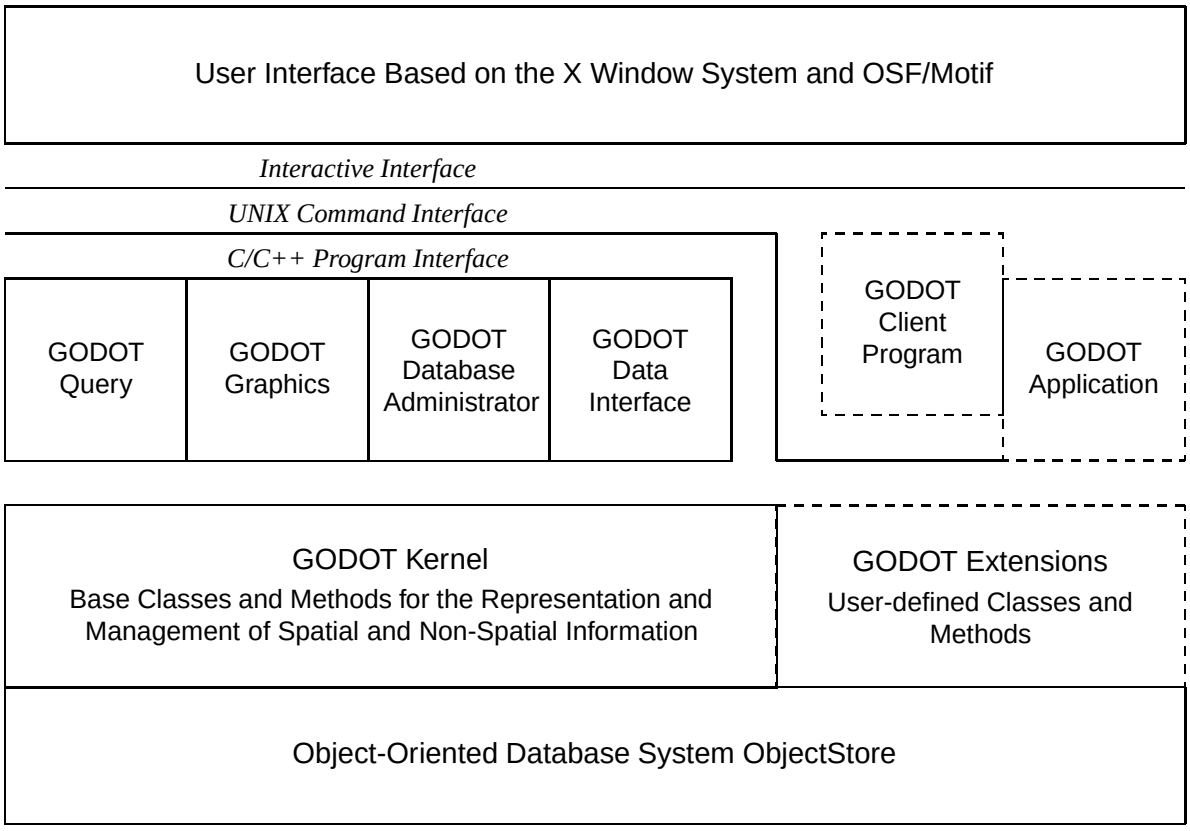


Figure 1: GODOT Architecture

The query component is tightly linked to a *graphics component* that manages the graphical representation of the geographic information. In particular, it is common to use the graphics component to visualize the result of user queries. On the other hand, one can use the graphics component to specify parts of a query by pointing to certain objects on the screen. This includes the ability to update geographic information by manipulating the corresponding graphic objects interactively.

The *database administrator component* provides the usual features for database schema manipulation, user administration, and system support. Once again, we can use some of ObjectStore's functionalities directly, while others need to be adapted to manage spatial data efficiently.

A major issue in GIS is the exchange of geometric data encoded in different formats. GODOT supports the integration of *data interfaces* as base components; these can be activated through any of the interface modules described in the following section. GODOT also has its own external data format, which is a subset of C++, encoded in ASCII. This external data format can be read easily by other systems. The execution of the code leads directly to the generation of the corresponding object classes and instances in the given database.

2.3 User Interfaces

One of GODOT's core functionalities is to be a GIS data server for a diverse and distributed collection of applications. For this purpose, GODOT offers a variety of client-server style interfaces. An *interactive interface* under the X Window System gives high-level graphical access to GODOT, especially for the occasional or non-expert user. A different kind of access is provided by the *UNIX command interface*, where GODOT queries

can be formulated by means of specialized commands that form an extension of the UNIX shell. Command procedures can then be implemented as shell scripts. Finally, a *C/C++ program interface* makes the GODOT modules available as a program library. This interface is typically used for more complex GODOT applications; it also allows remote access via remote procedure calls. For the implementation of the C/C++ program interface, the recommendations of the Object Management Group with regard to the Object Request Broker [4] will be taken into account.

3 Data Model

The GODOT data model is partitioned into three categories of objects:

1. *Thematic objects* are used to represent real-world objects. An important subcategory of thematic objects are the *geographic objects* or *geo-objects*. A thematic object is a geo-object if it is geometric in nature, i.e., if it has a spatial extension.

2. *Geometric objects* or *geometries* are used to describe the geometric features of geo-objects.

3. *Graphic objects* are used for the display of thematic objects. *Cartographic objects* are an important subcategory of graphic objects.

The various relationships between these categories are shown in Figure 2, using a simple example. Note that we use strings of type X<Y> as object identifiers, where X denotes the class of the object. There are three thematic objects in that example: Two geo-objects City<Ulm> and City<Neu-Ulm> to represent the twin cities Ulm and Neu-Ulm, and a simple thematic object CoordCommittee<31> to represent the coordination committee of the two cities. The two cities, respectively their corresponding geo-objects, are connected to geometric objects to represent their shapes and to several graphic objects for their cartographic representation.

In the sequel, the three object categories are discussed in turn.

3.1 Thematic Objects and Geo-Objects

In the GODOT data model, geographic and environmental information is represented by so-called *thematic objects*. Thematic objects may be simple or complex, i.e., composed of several other thematic objects. Examples of thematic objects include the coordination committee and the cities in Figure 2, or a species in a natural resource information system.

Thematic objects may have different kinds of attributes to represent a variety of geographic and environmental features:

1. Attributes of an elementary type (e.g., text strings or real numbers)

2. Attributes of a complex type (e.g., embedded classes in C++)

3. Attributes of a referential type (e.g., pointers to other thematic objects)

The most important subset of the thematic objects is formed by the *geo-objects*, which are characterized by an attribute that is a geometric object. A geo-object therefore has a location and a spatial extension. If a geo-object is complex, a number of integrity constraints need to be enforced. For example, the geometry of a complex geo-object has to be the union of its component geometries.

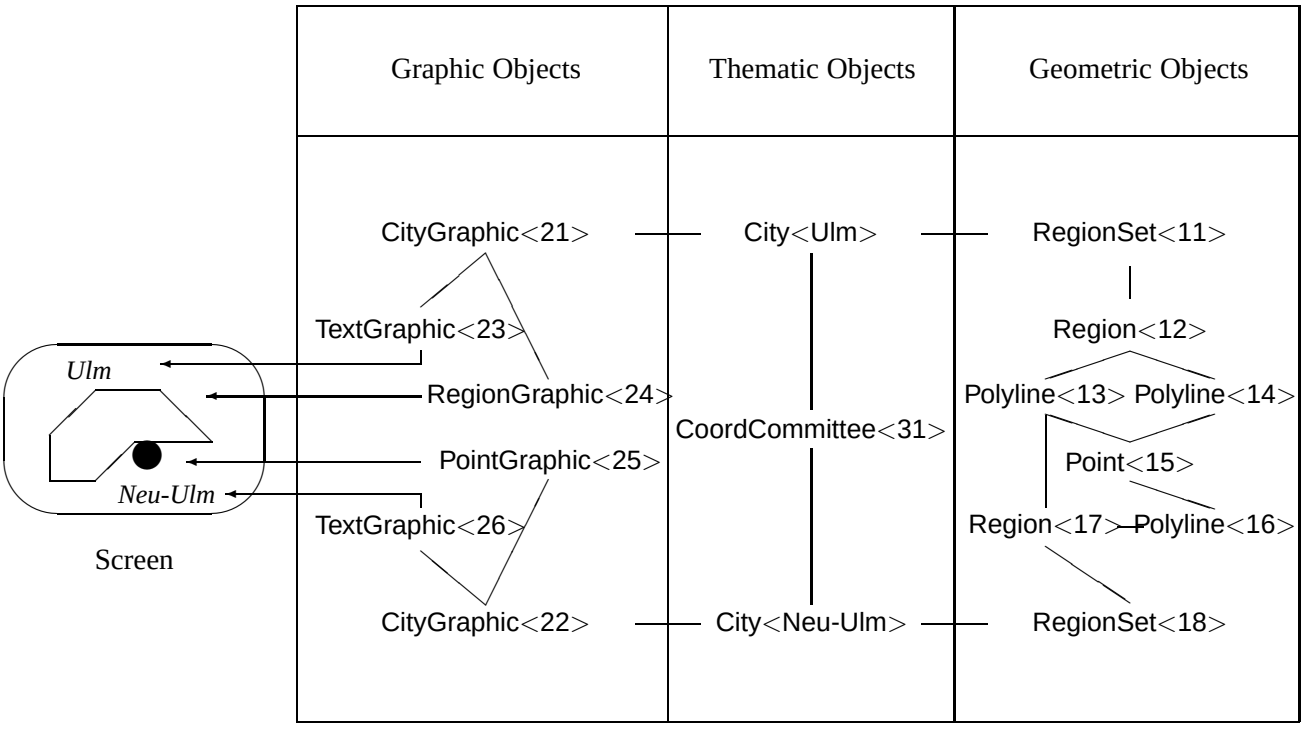


Figure 2: The three categories of the GODOT data model

3.2 Geometric Objects

Associated with each geo-object is a *geometric object*, which is typically composed of elementary geometric objects. These elementary geometric objects form the class *Geometry* with its three subclasses *Region*, *Arc*, and *Point*. An arc may be curved although the current implementation is restricted to (piecewise linear) polylines. Between these classes there exist the usual geometric relationships: A region has several bounding arcs. An arc may in turn belong to any number (including zero) of regions. Similarly, an arc has two endpoints, and any point may be the endpoint of any number of arcs.

Note that a geometric object can either be a singular point, arc, or region, or a collection of such singulars. If these singulars all belong to the class *Region* then the resulting complex object belongs to the class *RegionSet*. The classes *ArcSet* and *PointSet* are defined analogously. However, if the collection of singulars is heterogenous in the sense that it contains singulars of different types, it belongs to the class *GeometrySet*, which is the superclass of *RegionSet*, *ArcSet*, and *PointSet*.

This design guarantees that the result of a boolean set operation on two geometric objects can always be represented by exactly one geometric object. It also enables us to represent the geometries of any geo-object, however oddly shaped, with just one geometric object. Consider, for example, a river whose width varies widely, such that its geometry in the chosen accuracy is partly arc, partly region. In GODOT, the geometry of this river would be modeled by an instance of the class *GeometrySet*. Another example is a country whose area consists of several disconnected regions (e.g., the USA). This area would be represented by an instance of *RegionSet*.

3.3 Graphic Objects

Graphic objects are used for the (interactive or printed) display and the interactive update of thematic objects, in particular of geo-objects. The looks of a graphic object are defined in detail by its attributes, such as color, line width, or text font. Possible graphic representations include business graphics, tables, videos, raster images, or GIS-typical vector graphics. A thematic object can be linked to several graphic representations (e.g., for multiple scale display).

An important subcategory of graphic objects is formed by the *cartographic objects*, which are used for the graphic display of geo-objects. With this design, GODOT implies a clear separation between geographic and cartographic information. Cartographic objects contain methods that determine how the properties of a given geo-object are represented in terms of the graphics available. In particular, questions of scale and cartographic generalization are handled at this level and *not* at the level of thematic objects.

4 Outlook

In this short paper we sketched the basic architecture of the GODOT object-oriented GIS. We have finished the implementation of our data model on top of ObjectStore, and we are currently working on the graphic user interface. The first complete prototype should be functional no later than the end of 1993.

References

[1] M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, and S. Zdonik, The Object-Oriented Database System Manifesto, in *Proc. First Int. Conf. on Deductive and Object-Oriented Databases*, Kyoto, 1989.

[2] C. Lamb, G. Landis, J. Orenstein, and D. Weinreb, The ObjectStore Database System, *Communications of the ACM*, 34(10):50–63, October 1991.

[3] J. Orenstein and F. Manola. PROBE Spatial Data Modeling and Query Processing in an Image Database Application, *IEEE Transactions on Software Engineering*, 14(5):611–629, May 1988.

[4] Object Management Group, *The Common Object Request Broker: Architecture and Specification*, OMG, Framingham, Mass., 1992.

[5] P. Oosterom and T. Vijlbrief, Building a GIS on Top of the Open DBMS POSTGRES, in *Proc. EGIS'91*, Brussels, April 1991.

[6] M. Scholl and A. Voisard, Object-Oriented Database Systems for Geographic Applications: An Experiment With O_2, in: F. Bancilhon, C. Delobel and P. Kanellakis (Eds.), *The O_2 Book*, Morgan Kaufmann, San Mateo, Calif., 1992.

[7] H.-J. Schek and A. Wolf, From Extensible Databases to Interoperability Between Multiple Databases and GIS Applications, in: D. Abel and B. C. Ooi (Eds.), *Advances in Spatial Databases*, Springer, Berlin, 1993.

[8] M. Stonebraker, The Sequoia 2000 Project, in: D. Abel and B. C. Ooi (Eds.), *Advances in Spatial Databases*, Springer, Berlin, 1993.

Efficient Spatial Query Processing in Geographic Database Systems

Hans-Peter Kriegel, Thomas Brinkhoff, and Ralf Schneider

Institute for Computer Science, University of Munich
Leopoldstr. 11 B, D-80802 München, Germany
e-mail: {kriegel,brink,ralf}@dbs.informatik.uni-muenchen.de

1 Introduction

The management of spatial data in *geographic database systems* gained increasing importance during the last decade. Due to the high complexity of objects and queries and also due to extremely large data volumes, geographic database systems impose stringent requirements on their storage and access architecture with respect to efficient spatial query processing.

Geographic database systems are used in very different application environments. Therefore, it is not possible to find a compact set of operations fulfilling all requirements of geographic applications. But as described in [BHKS 93], *spatial selections* are of great importance within the set of spatial queries and operations. They do not only represent an own query class, but also serve as a very important basis for the operations such as the nearest neighbor query and the spatial join. Therefore, an efficient implementation of spatial selections is an important requirement for good overall performance of the complete geographic database system. The most frequent spatial selection is the *window query*: Given a rectilinear rectangle *W* of arbitrary size and a set of objects *M*, the window query yields all the objects of *M* intersecting *W*.

For the efficient processing of spatial selections, we propose a *spatial query processor* (see Figure 1). Its major goal is to reduce expensive steps by preprocessing operations in the preceding steps which reduce the number of objects investigated in an expensive step. In Figure 1 expensive steps are marked with the $-symbol.

A spatial selection is abstractly executed as a sequence of steps: First, we scale down the search space by *spatial indexing*. A spatial index organizes sets of tuples on secondary storage. Each tuple corresponds to a spatial object. It consists of a geometric key, an object identifier (ID) and a reference to the exact object geometry. Due to the arbitrary complexity of real geographic objects, it is not advisable to build up an index using the exact geometric description of the objects as a key; instead, simple approximations of the objects are used. Thus, the spatial index is not able to yield the exact result of a query. However, spatial indexing filters out a high number of objects not fulfilling the query.

Objects identified by the spatial index may fulfill the query. Therefore, we must inspect these *candidates* using a *geometric filter* which tests the geometric key or further approximations of the object geometry against the query condition. As a result, we obtain three classes of objects: *hits* fulfilling the query, *false hits* not fulfilling the query and *candidates* possibly fulfilling the query.

If we are only interested in the object identifiers, these hits are a subset of the set of answers (indicated by a dotted arrow in Figure 1). Only the candidates have to be transferred into main memory for further processing (indicated by a fat arrow in Figure 1). If the complete object geometry is required as an answer to the query, we have to *transfer the exact geometry* of hits *and* candidates into main memory.

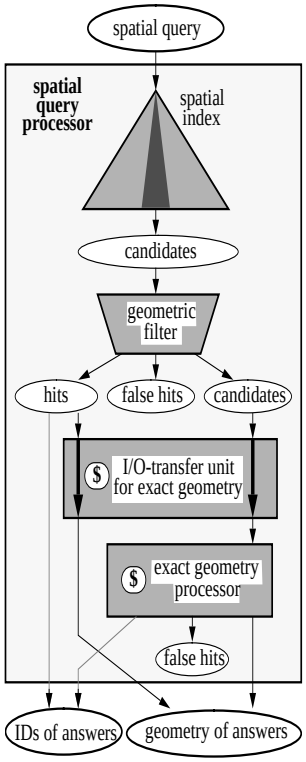


Figure 1: Spatial query processor

The transfer of the exact geometry may be very expensive, because the exact representations of objects can be large (compare e.g. [BHKS 93]) and, additionally for window queries, a high number of spatially adjacent objects has to be transferred. Therefore, a physically contiguous storage of spatially adjacent objects combined with a fast set-oriented I/O [Wei 89] is necessary to support large window queries.

The other expensive step is *processing the exact geometry* of an object. After filtering, the remaining candidates are investigated using complex computational geometry algorithms. Thus, it is decided whether a candidate fulfills the query or not.

The goal of this paper is to present techniques for efficiently supporting spatial selections using the above spatial query processor. First, in section 2 we discuss spatial access methods for scaling down the search space efficiently. In section 3, geometric filtering techniques using various approximations are proposed. Next, the efficient transfer of the exact geometry is discussed in section 4. Processing the exact geometry supported by decompositions is the topic of section 5. The paper concludes with a presentation of a concrete spatial query processor and suggestions for future work. We would like to emphasize that all the presented techniques have been implemented and tested with real cartography data.

2 Scaling down the search space

Considering spatial selections in more detail, it turns out that in general only a small and locally restricted part of the complete search space needs to be investigated. For an efficient scaling down of the search space, it is essential to use *spatial access methods (SAMs)* because a high number of spatial objects has to be organized. Access methods as an ingredient of the internal level of a database system partition the data space dynamically into *regions* that correspond to pages on secondary storage. One-dimensional access methods like B-trees or linear hashing schemes are not suitable for geographic database systems because they do not organize the spatial objects with respect to their location and extension in the two-dimensional data space. Because of the arbitrary complexity of spatial objects, SAMs for simpler spatial objects such as points or rectangles are widely discussed in the literature, e.g. the grid file, the quadtree, the buddy-tree, and the R-tree. Samet provides an excellent survey [Sam 90] of almost all of these methods.

Simply stated, SAMs for extended spatial objects use one of three techniques: The *clipping technique* partitions the data space into disjoint regions. The objects are associated with each of the regions they intersect. In general, the clipping technique leads to poor query performance, since the objects may be stored in multiple pages. The *transformation technique* views an object as a point in some higher-dimensional parameter space. Since transformations do not preserve the spatial neighborhood of objects in the original data space, and since the distribution of parameter points is extremely skewed, the query performance tends to be quite bad. The *overlapping regions technique* assigns each object to exactly one region. However, the regions may overlap. Consequently, there may exist several regions potentially containing the object searched.

Performance comparisons (e.g. [HS 92]) demonstrate that the well-known SAMs do not significantly differ in performance. We favor the *R*-tree* [BKSS 90], an improved variant of the R-tree, because it has proven to be simple, robust, and efficient. The R*-tree uses the technique of overlapping regions and demonstrates that it is possible to organize spatial objects with an extremely small overlap of the regions. From our point

of view, integrating SAMs into geographic database systems is indispensable for query processing, but further research in SAMs will improve the overall performance only marginally. Therefore, additional concepts have to be integrated to accelerate spatial query processing.

3 Geometric filtering by approximations

As described above, spatial objects are organized and accessed by SAMs using geometric keys that represent the most important features of the objects (location and extension). For extended spatial objects the *minimum bounding box (BB)* is the most popular geometric key. Using the BB, the complexity of an object is reduced to four parameters. Using BBs provides a fast but inaccurate filter for the response set. The more the area of the BB differs from the area of the original object, the less accurate geometric filtering is, i.e. the candidate set includes a lot of false hits.

In order to get expressive and realistic results on the approximation quality of BBs, we investigated polygonal objects of various real maps. To be as general as possible, we used maps from different sources with different resolutions. The data files contain natural objects such as islands and lakes, and administrative areas such as counties.

map	*min*	*max*	*fa*
Europe	0.25	21.14	0.93
BW	0.20	5.01	0.93
Lakes	0.21	22.11	0.97
Afrika	0.34	5.64	0.89

Table 1: False area of the BB

The accuracy of the filtering step is maximized by minimizing the deviation of the approximation from the original object. We measure this deviation by the *false area* of the approximation normalized to the area of the original object.

Table 1 shows impressively that real cartography objects are only roughly approximated by BBs. In this table, *fa* is the average false area; *min* and *max* denote the minimum and the maximum false area in the map, respectively.

This investigation was the starting point to look for other approximations that have better quality than the BB. Additionally, they should be simple to provide a fast filter. These requirements are fulfilled by convex shapes with straight-line borders. Therefore, we tested the *rotated bounding box (RBB)*, the *convex hull (CH)*, the minimum enclosing convex *4-corner (4-C)*, and the *5-corner (5-C)*.

Figure 2: Approximations with their number of parameters (#p) and average false area (fa)

In [BKS 93], we measured the approximation qualities in a detailed empirical investigation. As expected, the more parameters are used for the representation of an approximation, the better the approximation quality is. Naturally, the convex hull has the best approximation quality. However, the storage requirement of convex hulls has extreme variations and on the average is much higher than the storage requirements of the other approximations. In summary, we conclude that the 5-corner yields the best trade-off between additional storage requirement and improvement of approximation quality. The 6 additional parameters compared to the BB pay off by reducing the average false area by 65% compared to the BB.

Integrating the 5-corner in geometric filtering can be done in two different ways. First, the common BB remains the geometric key and the 5-corner is *additionally* stored. This approach can be applied to all known SAMs based on BBs. In the second approach, the 5-corner is used as geometric key *instead of* the BB which

saves storage but is not suitable for the transformation technique. In [BKS 93] we have shown that this approach can be efficiently realized using the R*-tree.

4 Transfer of the geometry into main memory

Most SAMs proposed up to now store either the approximations or the exact geometry of a small number objects in their data pages. However, the pages storing the geometry of spatially adjacent objects are distributed arbitrarily over the secondary storage. Typical window queries require the contents of many pages to be retrieved from the database. If these pages are arbitrarily distributed, we always need one seek operation for each accessed page. Because seeks are very expensive I/O operations, large window queries become extremely expensive. The physically contiguous storage of spatially adjacent objects combined with set I/O saves many seek operations.

The main question is how to determine suitable sets of pages contiguously stored on secondary storage and how to organize these sets dynamically. Our approach, the *scene organization*, stores spatially adjacent objects in a set of physically contiguous pages. Such a set corresponds to a subtree of a slightly modified R*-tree and is called *scene*. A scene is described by a minimum bounding box which is stored in a upper level of the R*-tree. The geometric keys and additional approximations are organized as described before. The scene organization allows dynamic changes of the database, assures a maximum scene size, and strives for a stable average scene size and high storage utilization. An algorithmic description is given in [BHKS 93].

The scene organization supports large window queries as well as small queries. A window query is processed by first determining all scenes that intersect the query window. If the degree of overlap between the scene and the query window is smaller than a query threshold determined experimentally, the window query is processed using the mechanism depicted in the left part of the diagram in Figure 3. Otherwise, the scene is completely transferred into main memory using the fast set I/O. Unfortunately, a scene may contain a number of false hits causing unnecessary transfer of objects into main memory. However, a relatively small number of false hits does not affect performance considerably, since the time for a transfer operation is much lower than the time for a seek. After transferring the scene into main memory, the query is processed as usual, however without further transfer after filtering. This sequence of steps is depicted in Figure 3.

In order to evaluate the scene organization, we carried out a detailed empirical performance comparison of the scene organization using real geographic data. The database consisted of 119,151 objects with an average size of 956 bytes. The page capacity was 4 kbyte. We tested 5 series of window queries with an area of the query window between 0.0625% and 16% of the data space. We weighed the cost for seeking a page on disk by a factor of 10 relative to the cost of a page transfer.

As expected, when the scene size increases, the seek cost decreases, and the transfer cost increases. The optimal scene size which leads to minimum access cost depends on the size of the queries. The larger the queries, the larger the optimal scene size is. However, this dependency is not very strong. Our tests show that if the size of the query window grows by a factor of 256, the optimal scene size grows only by a factor of 8. Additionally, the cost functions are rather flat in the proximity of their minimum. Thus, there exists a scene size which is almost optimal for all sizes of query windows.

We compared the performance of the scene organization to two conventional models:

- model 1: *storing the exact object geometry outside of the spatial access method*
- model 2: *storing the exact object geometry inside the data pages of the spatial access method*

In Table 2, the access costs of the three models are presented in terms of the speed-up factor for processing queries in comparison to model 1. Storing the exact object geometry inside the data pages (model 2) speeds up query processing by a factor of 2.3 to 2.9 in comparison to model 1, which stores the exact geometry outside of the data pages. The fundamental drawback is the fact that each access to the object geometry causes an additional seek operation. The scene organization is the clear winner of the performance comparison because

its clustering is not limited by the page size as in model 2. Even the processing of small window queries is performed considerably faster by the scene organization.

size of query windows (in % of data space):	0.0625%	0.25%	1%	4%	16%
geometry inside the data pages (model 2)	2.3	2.5	2.6	2.8	2.9
scene organization	6.5	11.7	18.6	25.4	30.6

Table 2: Speed-up factors for window queries in comparison to model 1

5 Exact investigation of the geometry

Geometric filtering is based on object approximation and therefore determines a set of *candidate objects* that may fulfil the query condition. The geometry processor tests whether a candidate object actually fulfils the query condition or not. This step is very time consuming and dominates the costs for spatial indexing and geometric filtering in many applications. Algorithms from the area of computational geometry are proposed to overcome this time bottleneck.

Due to the complexity of the objects on the one hand and the selectivity of spatial queries on the other hand, it is useful to *decompose the objects* into simpler components because the decomposition substitutes complex computational geometry algorithms by multiple calls of simpler and faster algorithms. The success of such processing depends on the ability to narrow down quickly the set of components that are affected by the spatial queries and operations.

In [SK 91] we showed how the decomposition approach can be used in order to improve the exact geometry processing of polygonal objects. In a preprocessing step, we decompose polygonal objects into a minimum set of disjoint trapezoids using the plane-sweep algorithm designed by Asano and Asano [AA 83]. Because we cannot define a complete spatial order on the set of trapezoids that are generated by this decomposition process, binary search on these trapezoids is not possible. Therefore, we propose to use the R*-tree for the spatial search. Due to its tree structure, the R*-tree permits nearly logarithmic searching in real applications. The R*-tree is designed as a SAM for secondary storage. In order to speed up the geometric test, we developed the *TR*-tree*, a variant of the R*-tree, designed to minimize the main memory operations and to store the trapezoids of the decomposed objects. The main characteristic of the structure of the TR*-tree is its small maximum number of entries per node. We obtain the best experimental results with a maximum number of 3 to 5. The TR*-tree representation of an accessed object is completely loaded into main memory for spatial query processing.

The performance of the TR*-tree cannot be analytically proven because the TR*-tree is a data structure that uses heuristic optimization strategies. Therefore, we compared the performance of the TR*-tree approach to plane-sweep algorithms in an experimental analysis. The results of this analysis documented that a speed-up factor of two orders of magnitude can be obtained when the intersection test of two polygonal objects is performed.

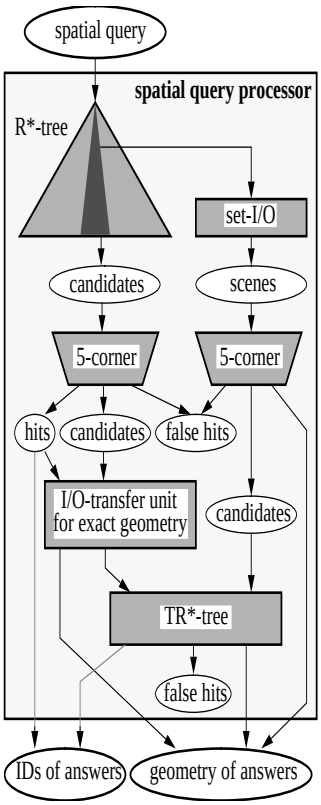


Figure 3: Spatial query processor

6 Summary and future work

In the previous sections, we presented a number of techniques for reducing query time. Figure 3 depicts our spatial query processor consisting of several building blocks using these techniques. The R*-tree is used as a simple, robust, and efficient spatial index. Our performance comparison of approximations demonstrates that the 5-corner is the best trade-off between additional storage requirement and improvement of approximation quality. Therefore, it is used as a geometric filter. In order to reduce the I/O cost for window queries retrieving the object geometry, we integrated the scene organization. If the degree of overlap between a scene and a query window is larger than a given query threshold, the complete scene is transferred into main memory using a fast set-I/O device. Otherwise, the R*-tree is completely traversed and the exact geometry of candidates (and if necessary of hits) is transferred after filtering. The exact geometry processor is realized by TR*-trees organizing decomposed objects.

In our future work, we want to apply the presented techniques for efficient processing of *spatial joins*. Furthermore, in order to increase the number of identified hits in the filtering step, it is necessary to integrate *kernel approximations* into our query processor. The application of the presented techniques to *3D-objects* is another important area of research activities.

References

[AA 83] Asano Ta. and Asano Te.: 'Minimum Partition of Polygonal Regions into Trapezoids,' Proc. 24th IEEE Annual Symp. on Foundations of Computer Science, 1983, pp. 233-241.

[BHKS 93] Brinkhoff T., Horn H., Kriegel H.-P., and Schneider R.: 'A Storage and Access Architecture for Efficient Query Processing in Spatial Database Systems,' Proc. 3rd Int. Symp. on Large Spatial Databases, Singapore, Lecture Notes in Computer Science, Vol. 692, Springer, 1993, pp. 357-376.

[BKS 93] Brinkhoff T., Kriegel H.-P., and Schneider R.: 'Comparison of Approximations of Complex Objects used for Approximation-based Query Processing in Spatial Database Systems,' Proc. 9th Int. Conf. on Data Engineering, Vienna, Austria, 1993, pp. 40-49.

[BKSS 90] Beckmann N., Kriegel H.-P., Schneider R., and Seeger B.: 'The R*-tree: An Efficient and Robust Access Method for Points and Rectangles,' Proc. ACM SIGMOD Int. Conf. on Management of Data, Atlantic City, NJ, 1990, pp. 322-331.

[HS 92] Hoel E.G. and Samet H.: 'A Qualitative Comparison Study of Data Structures for Large Line Segment Databases,' Proc. ACM SIGMOD Int. Conf. on Management of Data, San Diego, CA, 1992, pp. 205-214.

[Sam 90] Samet H.: 'The Design and Analysis of Spatial Data Structures,' Addison-Wesley, 1990.

[SK 91] Schneider R. and Kriegel H.-P.: 'The TR*-tree: A New Representation of Polygonal Objects Supporting Spatial Queries and Operations,' Proc. 7th Workshop on Computational Geometry, Bern, Switzerland, Lecture Notes in Computer Science, Vol. 553, Springer, 1991, pp. 249-264.

[Wei 89] Weikum G.: 'Set-Oriented Disk Access to Large Complex Objects,' Proc. 5th Int. Conf. on Data Engineering, Los Angeles, CA, 1989, pp. 426-433.

Spatial Indexing: Past and Future

Hongjun Lu *Beng-Chin Ooi*
Department of Information Systems and Computer Science
National University of Singapore
10 Kent Ridge Crescent, Singapore 0511
Internet: {luhj, ooibc}@iscs.nus.sg

Abstract

Efficient processing of queries in spatial database systems relies upon auxiliary indexing structures. A large number of spatial indexing structures have been proposed. In this short paper, we examine the basic issues in indexing spatial data, classify the existing indexing mechanisms according to their underlying data structures and the techniques used to handle non-zero sized objects, and briefly discuss future research directions.

1 Introduction

Spatial information processing has been a focus of research during the past decade for effective support for applications in computer vision, computer-aided design, solid modeling, geographic information systems (GIS), computational geometry, and etc. In spatial databases, data are associated with spatial coordinates and extents, and are retrieved based on spatial proximity. Spatial data consist of points, lines, polygons, volumes, and etc. Operators supported for spatial data such as geometric operators (eg., rotation and translation) and spatial operators (eg., spatial intersect and contain) are much harder to compute. Efficient processing of queries manipulating these relationships relies upon auxiliary indexing structures.

A record having k attributes can be viewed as a point in a k-dimensional space. Multi-attribute indexing has also been studied in the context of relational database systems. Indexing in spatial databases, however, is different from indexing in a conventional database in that data in a spatial database system are associated with coordinates in the Euclidian space and often represent *non-zero sized* objects. One of the major issues here is to effectively extend the existing indexing techniques for point (zero-sized) data to handle non-zero sized data. The purpose of indexing is to accelerate query processing. Queries in conventional databases can be divided into two basic types: *exact match queries* and *range queries* . In spatial databases, the range query is generalized to intersection search in which a search condition specifies a region and the results include all the objects that intersect with it. One factor that makes search in spatial database more complex is that the search is often not based on attribute values, but rather based on spatial properties of objects.

In addition to efficient support of both exact match and range queries, those desired properties for conventional indexing structures, such as high space utilization, small ratio between the sizes of index and data, etc. are still the design goals of spatial indexes. However, the same goals are more difficult to achieve for spatial indexes since the volume of data in a spatial database is usually much larger than that in a conventional database, and the distribution of data varies more drastically. Robustness, the ability of maintaining expected performance of an index mechanism over a wide range of data distribution and query patterns, becomes more important for spatial access methods.

A formidable number of spatial indexes have been proposed in the literature. In Section 2, we will categorize them in the hope of better understanding of their fundamental strengths and weeknesses. In Section 3 we will

speculate on research directions that might lead to some fruitful results. Due to limited space, our presentation is inevitably sketchy and incomplete. Interested readers are suggested to follow the pointers available through the paper's references.

2 Spatial indexing techniques

Though the spatial indexes are more complex compared to non-spatial indexes, they are developed from the well known basic indexing structures such as sorted arrays, binary trees, B-trees, and hashing, etc. Figure 1 shows the evolution of the spatial indexing structures from the underlying base structures. Solid arrows in the diagram indicate the relationships between a new structure and the original structures that it is based upon. The dashed arrows are used to indicate a relationship between a new structure and the structures from which the techniques used in the new structure originated, even though some were proposed independent of the others.

The underlying data structure of a spatial index determines the basic behavior of an index to a great extent. For example, the grid files [NHS84] are based on hashing so that the number of disk accesses required for exact match queries is expected to be constant. If the directory is stored as an array and all grid cells are of the same size, only two disk accesses are required to retrieve an object: one for the directory entry and another for the data page. On the other hand, as all hashing based-schemes, data skewness will lower the space utilization and increase the size of directory drastically, which leads to the multi-level tree structured directory [WhK85]. Similarly, all tree-based indexes share some common characteristics.

Besides the underlying data structure, another important factor that determines the performance of a spatial index is the way that it extends the basic structure to handle non-zero sized spatial objects. A basic approach of indexing spatial data in a space is to (recursively) partition the space into manageable number of smaller subspaces. Proper handling of the data objects and the partitioning process becomes a critical issue to extend the basic structure to index non-zero sized objects. Major techniques for extension can be categorized into the following classes [SeK88]:

- *Object Mapping (Transformation)* : This approach maps objects from a k-dimensional space into points in a $2k$-dimensional space. Alternatively, objects are mapped from k-dimensional space into points in a linear space, which can be indexed using any basic data structure for point data.
- *Object Duplication/Clipping*: The object duplication approach associates each object with an identifier and duplicate it in all subspaces that the object intersects. The object clipping method decomposes an object into smaller objects such that each component is totally included in one subspace.
- *Object Bounding*: Under this mechanism, the partitioning of the subspace is not arbitrary. Certain measures are supported to maintain that each data object is totally included in a subspace.

Table 1 groups various index structures according to the techniques used to handle non-zero sized spatial objects.

Each of the above approaches has its own strengths and weaknesses, which directly affect the performance of indexes using it. For example, the major advantage of object mapping is that, after mapping, non-zero sized objects can be treated as points in a k-dimensional space and proven techniques can be applied. However, after mapping, the spatial relationship between the objects in the original space may not be preserved in the mapped space. As a consequence, even if the original data are clustered, the mapped data may not be, making the intersection search slow. Furthermore, not all queries can be meaningfully transformed. The major advantage of object duplication and clipping is that the data structures used are straight forward extensions of the underlying indexing structures. Also, both points and multi-dimensional non-zero sized objects can be stored together in one file without having to modify the structure. The disadvantage is that duplication of object identifiers increases the cost of storage and updates. As for the object bounding method, algorithms for maintaining effective enclosing subspaces of objects can be complex and expensive. They may also require extra storage for the directory structures and rendersing the search expensive.

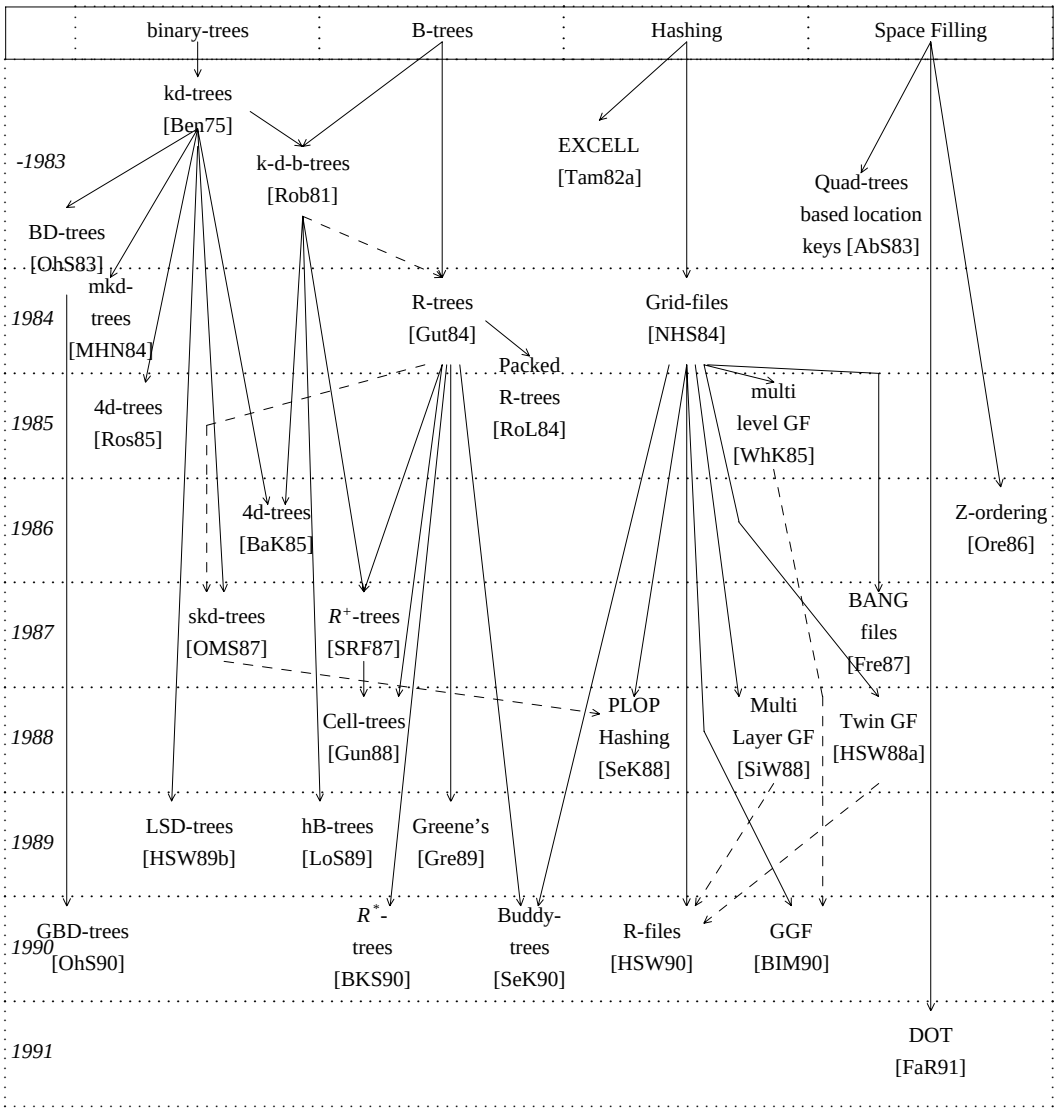


Figure 1: Evolution of spatial index structures.

Object Mapping		*Object Duplication/Clipping*	*Object Bounding*
k-d to 2k-d	*k-d to 1-d*		
Grid files	locational keys	Grid files	multi-level grid files
Twin grid files	Z-ordering	EXCELL	R-files
BANG files	DOT	mkd-trees	PLOP-hashing
GGF		R^+-trees	skd-trees
EXCELL		Cell trees	GBD-trees
K-D-B-trees		BANG files	R-trees
4d-trees			Packed R-trees
hB-trees			R^*-trees
LSD-trees			Buddy-trees

Table 1: Classification of spatial index structures.

3 Discussions

It can be seen that there are far more spatial access methods that have been proposed compared to the number of access methods supported in relational database systems. This might be unavoidable as the applications using spatial data are more diversified. It also indicates that most of them may not be satisfactory in one way or another. While more specialized access methods can be devised to yield good performance for certain data distribution and queries, it seems more important to develop a sound understanding of behavior of a few robust access methods under varying conditions [GuB90]. As such, we could identify one or two indexing structures, for each particular application area such as CAD/CAM and GIS, so that satisfactory performance can be maintained or at least, the worst-case performance could be predicted.

To reduce the number of indexing techniques to be supported in a system whose applications may have different requirements, extensibility is one of the major design goals for new spatial indexing structures. An index strudture can be made extensible by supporting variants of its basic structure. For example, the experimental results show that, although the leaf layer of an skd-tree [OMS87] can reduce disk page accesses by limiting the bounding rectangle coverage, it will increase them drastically due to the size of index when data are skewed [OSM91]. With this observation, an extensible skd-tree index can be implemented that supports both structures, with or without the leaf layer. The users can choose one of them based on the data skewness in their applications. Thus, for applications where data is highly skewed, the leaf layer will not be built. Another alternative is to include a skew detection mechanism in the insertion algorithms. When high data skewness is detected, the index will re-structured and stop to build the leaf layer. Furthermore, a number of proposed indexing mechanisms with the same basic underlying data structure (e.g. tree) can be integrated as one extensible index that can maintain satisfactory performance over a wider range of applications. In this case, a set of generic functions such as data space partitioning, object insertion and deletion can be defined based on the underlying data structure. Different implementations of those functions, including user defined ones, are stored in a linkable library. By instantiating selected implemenations, a near-optimal indexing structure for a particular application can be generated.

Both identifying the robust access methods and designing effective extensible spatial indexes require us to develop some benchmark databases and queries. So far, most comparative studies used synthetic databases consisting of limited sets of spatial data objects that are generated using random number generators with desired distributions and skewness with few exceptions[HoS92]. Although such randomly generated databases may maintain certain fairness for the comparison, they may not reflect the characteristics of the real world data. The ideal benchmark should be *abstracted from* the typical applications. We hope that some acceptable benchmarks for spatial database systems will be available in the near future.

Indexing is a low level service provided to improve the overall system performance. Thus, it is important to develop indexing techniques taking the whole system into account. So far, most studies on spatial indexing focus on the data structures and algorithms; less work has been done on the interaction between index structures and other closely related system components, such as query optimizers, system catalogues, buffer managers. In addition to those well known issues such as concurrent updating and on-line construction of large indexed, effective buffer management, one special issue here is the integration of spatial and non-spatial data. With a good acceptance of object-oriented paradigm in multi-media systems, the coupling of spatial and non-spatial data becomes tighter. Range queries with both spatial (geometric) and non-spatial conditions are not rare. To provide reasonable performance for such queries, an integrated approach is suggested where the non-spatial attributes to be requested flow directly into the overall data structure [Ohle92]. In a recent paper, Pagel, Six and Toben proposed an area-sensitive transformation technique to address this problem [PST93]. While such a technique provides a solution, it may not be feasible to build such index for each of the frequently accessed non-spatial attributes. Some more general built-in mechanism in spatial indexes are needed.

Finally, we would like to further emphasize that, solutions to all above mentioned issues require close and effective collaborations between the computer scientists and the application developers. High performance indexing techniques can only be developed with a thorough understanding of the usage of spatial data, including

the access patterns and the post processing after data are brought into memory. At the same time, application developers may be able to provide certain services or tune their algorithms to avoid some of the limitations of underlying indexing mechanisms.

References

[AbS83] D. J. Abel and J. L. Smith, A data structure and algorithm based on a linear key for a rectangle retrieval problem, *Int. Journal of Comp. Vision, Graphics, and Image Processing,* 24(1), 1983, 1-13.

[BaK86] J. Banerjee and W. Kim, Supporting VLSI geometry operations in a database system, *Proc. 2nd Data Engineering Conf.,* Los Angeles, CA, Feb. 1986, 409-415.

[Ben75] J. L. Bentley, Multidimensional binary search trees used for associative searching, *Comm. ACM,* 18(9) 1975, 509-517.

[BIM90] H. Blanken, A. Ijbema, P. Meek and B. Akker, The generalized grid file: Description and performance aspects, *Proc. 6th Data Engineering Conf.,* Los Angeles, CA, Feb. 1990, 380-388.

[BKS90] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger, The R-tree: An efficient and robust access method for points and rectangles, *Proc. ACM SIGMOD Conf.,* Atlantic City, NJ, May 1990, 322-331.

[FaR91] C. Faloutsos and Y. Rong, DOT: A spatial access method using fractals, *Proc. 7th Data Engineering Conf.,,* Kobe, Japan, Apr. 1991, 152-159.

[Fre87] M. Freeston, The BANG file: A new kind of grid file, *Proc. ACM SIGMOD Conf.,* San Francisco, CA, May 1987, 260-269.

[Gre89] D. Greene, An implementation and performance analysis of spatial data access methods, *Proc. 5th Data Engineering Conf.,* Los Angeles, CA, Feb. 1989, 606-615.

[Gun88] O. Gunther, *Efficient structures for geometric data management,* Lecture Notes in Computer Science 337, Springer-Verlag, 1988.

[GuB90] O. Gunther and A. Buchemann, Research issues in spatial databases, *ACM SIGMOD Record,* 19(4), Dec. 1988, 61-68.

[Gut84] A. Guttman, R-trees: A dynamic index structure for spatial searching, *Proc. ACM SIGMOD Conf.,* Boston, MA, June 1984, 47-57.

[HSW88] A. Hutflesz, H.-W. Six, and P. Widmayer, The twin grid file: a nearly space optimal index structure, *Proc. Int. Conf. on Extending Database Technology,* Venice, Italy, Apr. 1988.

[HSW89] A. Henrich, H.-W. Six, and P. Widmayer, The LSD tree: spatial access to multidimensional point and non-point objects, *Proc. 15th VLDB Conf.,* Amsterdam, Aug. 1989.

[HoS92] E.G. Hoel and H.Samet, A qualitative comparison study of data structures for large line segment databases, *Proc. ACM SIGMOD Conf.,* San Diego, California, June 1992, 205-214.

[HSW90] A. Hutflesz, H.-W. Six, and P. Widmayer, The R-file: An efficient access structure for proximity queries, *Proc. 6th Data Engineering Conf.,* Los Angeles, CA, 1990, 372-379.

[LoS89] D. Lomet and B. Salzberg, The hB-tree: a robust multi-attribute search indexing method, *Proc. 5th Data Engineering Conf.,* Los Angeles, CA, Feb. 1989.

[MHN84] T. Matsuyama, L.V. Hao, and M. Nagao, A file organization for geographic information systems based on spatial proximity, *Int. Journal Comp. Vision, Graphics, and Image Processing,* 26(3), 1984, 303-318.

[NHS84] J. Nievergelt, H. Hinterberger, and K. C. Sevcik, The grid file: An adaptable, symmetric multikey file structure, *ACM Trans. on Database Sys.* 9(1), 1984, 38-71.

[Ohle92] T. Ohler, The multi class grid file: an access structure for multi class range queries, *Proc. 5th Int. Sym. on Spatial Data Handling,* Charleston, SC, Oct. 1992, 260-271.

[OhS83] Y. Ohsawa and M. Sakauchi, The BD-tree – a new n-dimensional data structure with highly efficient dynamic characteristics, *Proc. IFIP Congress*, Paris, North-Holland, 1983, 539-544.

[OhS90] Y. Ohsawa and M. Sakauchi, A new tree type data structure with homogeneous nodes suitable for a very large spatial database, *Proc. 6th Data Engineering Conf.*, Los Angeles, CA, Feb. 1990, 296-303.

[OMS87] B. C. Ooi, K. J. McDonell, and R. Sacks-Davis, Spatial kd-tree: An indexing mechanism for spatial databases, *Proc. IEEE Int. Comp. Software and Applications Conf.*, Kyoto, Japan, 1987.

[OSM91] B. C. Ooi, R. Sacks-Davis, and K. J. McDonell, Spatial indexing by binary decomposition and spatial bounding, *Information Systems*, 16(2), 211-237, 1991.

[Ore86] J. A. Orenstein, Spatial query processing in an object-oriented database system, *Proc. ACM SIGMOD Conf.*, Washington, D.C., May 1986, 326-336.

[PST93] B.-U. Pagel, H.-W. Six, and H. Toben, The transformation technique for spatial objects revisited, *Proc. 3rd Int. Sym. on Large Spatial Database Sys.*, Singapore, June 1993, 73-88.

[Rob81] J. T. Robinson, The K-D-B-tree: A search structure for large multidimensional dynamic indexes, *Proc. ACM SIGMOD Conf.*, Ann Arbor, Michigan, June 1981, 10-18.

[RoL85] N. Roussopoulos and D. Leifker, Direct spatial search on pictorial databases using packed R-trees, *Proc. ACM SIGMOD Conf.*, Austin, Texas, May 1985, 17-31.

[Ros85] J. B. Rosenberg, Geographical data structures compared: A study of data structures supporting region queries, *IEEE Trans. on Comp. Aided Design CAD*, 4(1), 53-67, 1985.

[SeK88] B. Seeger and H. Kriegel, Techniques for design and implementation of efficient spatial access methods, *Proc. 14th VLDB Conf.*, Los Angeles, CA, Aug. 1988, 360-371.

[SeK90] B. Seeger and H. Kriegel, The buddy-tree: An efficient and robust access method for spatial data base systems, *Proc. 16th VLDB Conf.*, Brisbane, Australia, Aug. 1990, 590-601.

[SRF87] T. Sellis, N. Roussopoulos, and C. Faloutsos, The R^+-tree: A dynamic index for multi-dimensional objects, *Proc. 13th VLDB Conf.*, Brighton, England, Sep. 1987, 507-518.

[SiW88] H. Six and P. Widmayer, Spatial searching in geometric databases, *Proc. 4th Data Engineering Conf.*, Los Angeles, CA, Feb. 1988, 496-503.

[Tam82] M. Tamminen, Efficient spatial access to a data base, *Proc. SIGMOD Conf.*, May 1982, 200-206.

[WhK85] K. Whang and R. Krishnamurthy, *Multilevel grid files*, Report RC 11516, IBM Thomas J. Watson Research Center, Yorktown Heights, New York, 1985. Also in *Proc. The 2nd Int. Sym. on Database Sys. for Advanced Applications*, Tokyo, Japan, April 1991, 449-459.

Interactive Spatial Directories

Brian C. Schmult, H. V. Jagadish, and S. Kicha Ganapathy
AT&T Bell Laboratories

Abstract

A spatial directory is a directory of places, and information, things and events associated with places, where classification and searching are based on location (or spatial proximity) as well as on the name or type of an item. The result of a query could be alphanumeric or multimedia information, or a map. In this paper we sketch some of the issues we are tackling in the Interactive Spatial Directory (ISD) project.

1 Overview

A typical entry in an interactive spatial directory might be for an auto service station. For the service station, there would be a record of its location, hours of service, prices for different grades of gasoline, and so forth. A typical query will be based on proximity ("an auto service station within five miles of where I am"). The typical query result will be a severe projection, since the database may often contain far more information about the service station than the user desires. Further interactive querying can elicit desired additional information, such as gasoline prices, or a map showing the location of the service station and directions to get there. In this manner, an ISD is functioning like an inverted yellow pages, where the user goes from map to information instead of from advertisement to map.

The ISD is a set of services, such as the interactive query-presentation loop described above, centered around a core system that includes a spatial database. The spatial database also integrates traditional searching based on name and classification, and stores alphanumeric, multimedia, and geometric information. The services will be network based, and some will require location determination, for instance, the "where I am" part of the query above. Location based services might be based on GPS (Global Positioning System) [Potmesil, 1993] for location determination.

The defining characteristic of the ISD is that it will be a large, seamless store of potentially conflicting data coming from many sources, that must be indexed both traditionally and spatially in real time by many users. The data set is expected to be too large to be economical to unify into a monolithic database since it comes in small pieces from many sources. This raises a number of problems, including representation, indexing, automatic location of inconsistencies, simplified billing, etc. In this paper we do not consider such issues as billing and location determination. Instead we focus on the database core.

This paper focuses on three major ISD issues. First is the data representations that accommodate heterogeneous and conflicting data. The second is spatial indexing. The third is getting adequate performance for real-time access by multiple users. These issues are addressed in Sections 3-5 respectively. First we desribe the overall architecture of our database.

2 Schema Design

We use an object-oriented database system, Ode [Biliris *et al.*, 1992], which is based on C++. There are four fundamental classes of interest: logical entities, physical (or geometric) entities, representations, and attached information. (See [Jagadish and O'Gorman, 1989] for a detailed study of the issues in modeling logical and physical entities.) A *logical entity* is an object with meaning to the user, such as a road or a building. A single logical entity may have multiple physical *representations* (from different providers, or on account of differences in scale, etc. – see below). Each representation comprises one or more *physical entities*, such as line segments and rectangles. Other information (i.e. hours of operation) may be included as a slot in a type, but many entities (logical and physical) will have much instance-specific data, resulting in the *attached information* class. This is similar to a property list, or a binary large object as used in relational databases.

The main components of the base logical entity are the names and aliases, the types, plus one level of governmental containment, such as which county a municipality is in. The containment is particularly useful for limiting certain types of queries. Geometry is represented by physical entities such as points, polylines, polygons, and polygons with holes. Topological representation [White, 1980] is supported (but not required) by placing left and right memberships as attached information on polylines. Grid data will also be supported, including elevations, images, and characteristics such as land use/land cover.

2.1 Types

The domain of a query in an object-oriented database is usually the set of all objects of a given type. The type hierarchy of C++, and hence Ode, supports multiple inheritance, and so an entry is made under the most restrictive type known for it. For instance, an auto service station may be a subtype of both a gas station and an auto repair shop. Queries specified on supertypes return objects that are instances of its subtypes as well. Thus, a query for gas stations in some vicinity would return gast stations as well as auto service stations in the vicinity.

With the passage of time, new types of logical entities will be required. This must be automatic, since each information provider specifies the type(s) used by it, and we expect information sources to be added, deleted and updated frequently. New types may be derived from existing ones or newly created. It is not possible to have all possible types pre-compiled; for instance, someone could own a combination gas station and flower shop. As far as the Ode database is concerned, the creation of new types does not affect any existing data. When new objects, instances of the new type, are created, appropriate database catalog entries are made for the type, transparent to the application or database user. The challenge we are attempting to solve is for applications to automatically link in type-related code on the fly, so that recompilation of the world is not required for each new type.

Users may often not be aware of the different types defined in the schema. A user unaware of type names can ask a proximity query and retrieve all objects within the specified area, independent of type. Applications may query objects to determine their type, when required. A built-in function to determine the type of an object has been proposed for inclusion in C++, but is not currently available. We use a virtual function `my_type_is()` as a work-around for the present. This virtual function is defined for the class logical entity and all its subclasses. For this to work, all individual logical types must be derived, directly or indirectly, from the base type logical entity. `my_type_is()` returns the primary type. The entity constructors also automatically build a list of all types, represented by code numbers, for interrogation by applications.

3 Heterogeneity

It is expected that data in the spatial directory is obtained from multiple sources. Several problems of heterogeneity are created as a result.

First, the all important location attributes may not always be defined in the same way. We fix this by translating all location coordinates to latitude longitude pairs and using these as a standard for internal representation.

User queries also have all user-specified location attributes translated into the latitude longitude pair representation.

Often, point locations are specified relative to benchmarks, rather than in absolute terms, both in the data supplied and in the queries. It is difficult to compare data specified absolutely with data specified relatively. It is even harder to compare data points specified relative to different benchmarks. Our solution is to convert all locations to absolute values. Such conversion causes no problem in the query. In the database, however, the reference benchmark may itself move at times, so a hard conversion from relative to absolute coordinates is not appropriate. Instead, we use materialized views, retaining the definition of the coordinates of a point relative to a benchmark, but presomputing and storing the absolute value, and updating it with a trigger when the reference point is moved.

The next problem is distinct type names being assigned for essentially the same data obtained from different providers. Our "solution" is to treat these as distinct unrelated data, passing the buck to the user or a higher level application. These different data sets would have distinct index structures associated with them. Any combination of data sets could be searched in response to a user query. When a query specifies all variants of a conceptual type, hits from the various data sets are combined seamlessly.

Even worse, real data is often inconsistent. Thus, there may be conflicting place names, outright errors in names, missing data, and multiple versions of the same point or line from different sources that do not line up. The system scale, and our unwillingness to do special case hand-editing of inconsistencies, prohibits general unification. Instead, inconsistencies are exposed to the user by storing separately the information available from each provider, and making available to the user the version preferred for a particular application.

Similarly, map information is often available at different scales. This is frequently necessary, as displays at various scales require corresponding data. Work has been done to simplify detailed representations automatically [Whyatt and Wade, 1988], but this is not always possible, as some seemingly minor quirks may have legal significance, requiring that reduced resolution representations be explicitly created. Once more, it is preferable to show the user one or more distinct maps as may be appropriate for a particular application than to attempt reconciliation.

In short, the default technique to manage inconsistency is to have multiple logical entities, and let the application or user resolve the multiplicity. In addition, we plan to implement algorithms to search for objects that appear to be the same, based on a similarity measure between names, types, locations and geometry. For instance, see [Jagadish, 1991]. Where unification is impossible, an explicit relationship will be attached between the objects so that applications clearly know that there are multiple versions of real truth.

3.1 Representations

Each logical entity has one or more *representations*. Each representation, in turn, comprises one or more physical entities. (See [Story and Jagadish, 1991] for similar ideas in the context of document management.) For instance, a township is a logical entity. One representation of a township may show roads, another reprentation may show prominent buildings, and a third representation may show topography. Additional representations may present this information at a different scale. A road-map for a township may be presented as a set of line segments, each of which is a physical entity. Representations may also contain non-geometric information, such as road names or population figures.

Often, different representations of a logical entity will be derived from different sources. It is possible for representations to disagree on alphanumeric (population figure) or geometric (road location) data. Each representation is identified by some "header" information including an ID, the source of the information, the scale or resolution (stored as the minimum feature size), and the date range to which the information pertains. The date is particularly important for historical maps.

Spatial indices point to the physical entities (low-level geometry), so where multiple representations of a logical entity exist, a spatial query might return all representations. In some cases this is desired, but frequently

the application wants a unique representation. The query mechanism includes provision for a representation specification. The specification lists whether to use a default source, or which one is preferred or required; the range of permissible scales and whether this is preferred or required; and the permissible time range. The specification must be on a per-type basis, since a query might want geometry from one source and population from another. To make this more efficient for common cases, we have designed an encoding of representation descriptors, and combined it with an indexing technique described in the next paragraph.

It has been suggested, for object oriented databases, that it is sometimes effective to have indices over superclasses include instances of derived subclasses as well, with markers in the leaf nodes to indicate for each instance object which subclass it belonged to if any [Kim *et al.*, 1989]. We extend this idea to the case of multiple representations. An index structure on a particular object collection obtains the individual representations of the object instead of the object itself. Queries interested in all representations of an object can look them up; those interested only in a specific representation, can directly obtain the appropriate representation from the index structure.

4 Spatial Indexing

Performance of the ISD must be optimized for read access by many users, many of which will be interactive. Updates occur less frequently for most data. (There are some types of data that may change rapidly and require frequent updates, such as traffic or weather conditions. These exceptions must be treated specially). In other words, the cost trade-offs favor pre-computation where possible, and the maintenance of multiple index structures. In particular, spatial data is updated infrequently – rivers may change course, new buildings may be constructed, but such changes are typically on a much longer time scale than most queries. In fact, we anticipate that most spatial data updates can be applied in a batch mode at some low frequency, such as once a day.

We build one single global spatial index over all physical entities (lines, polygons, etc.) in the database. Given the large expected size of the database, this index will be huge. Since updates are batched and infrequent, we do not expect the large size to be a problem. On the other hand, having a single spatial index makes it easy to determine all objects within the vicinity of a specified point, and makes our default query plan, described below, optimal in most cases. Secondary spatial indices also exist, frequently associated with the data from each provider. These will be of particular use for specialized applications, such as hotel location and reservations.

Access by spatial indexing is completed through a set of backpointers. An index bucket points to a specific physical entity, which might be shared among logical entities, and/or might be duplicated by other representations of the same logical entity. (Physical entities may be shared, but not representations.) For instance a line may be part of a township boundary, a county boundary, as well as a state boundary, There is a backpointer from each physical entity to its containing representations, and a backpointer from each representation to its containing logical entity. When multiple representations are returned, the two levels of backpointers permit an application to throw out undesired physical entities.

When indexing is done by name, logical type, and/or other non-spatial attributes, the appropriate representation is selected based on its header, and a unique set of physical entities is obtained. Duplication is not possible and backpointers are not required.

A typical query will involve a region around a specified point. The global spatial index is used to identify all objects within the region of interest. Usually, the query will be interested not in all objects within the specified region of interest, but rather in objects of a particular (logical) type and possessing particular properties. These additional requirements are applied as a filter to the set of objects retrieved by the proximity query alone. (Our implementation relies on the fact that query regions for proximity queries are usually very small, thereby avoiding the need for a query optimizer).

Most providers are expected to provide map data in a *topological representation*, that is, as a collection of line segments. Where the data represents regions rather than lines or polylines, the line segments represent the

boundary of the region and the regions to the left and right of such boundary line segments are explicitly marked. Where there is an intersection of boundaries, line segments are terminated and the intersection point explicitly marked.

When data is obtained from multiple sources, such explicit marking of intersection is not possible. On the other hand, region intersections are likely to be the basis of many queries and hence gain importance. Boundaries of intersection regions have to be found by intersecting line segments of the individual boundaries. This has to be done at query time since data is from many different autonomous sources, and often one of the regions to be intersected may be a specified query region.

Efficient indexing of line segments is performed using the Hough transform space as suggested in [Jagadish, 1990]. The basic idea is that a line segment becomes a point in the transform space and can thus be retrieved efficiently using a point indexing technique. A region in the original space becomes a region in the transform space as well, and proximity notions are preserved.

5 Related Work

There are two types of similar work, Geographic Information Systems (GIS), and Advanced Travelers Information Services (ATIS). ATIS systems subsume directory services and street atlas systems.

GIS's are mainly concerned with manipulating geometric and thematic data related to maps. See [Vijlbrief and Oosterom, 1992] for a quick survey. The commercial ARC/INFO system [Morehouse, 1985] is representative. It uses topologically organized data split into separate maps (called coverages) and layers. There is no spatial index – each map is a bucket. Effort goes into seamlessly moving between maps. Our objective is to have only one map, and permit any region to be as easy to use as an independent GIS map.

ATIS's demonstrate the style of interaction we desire, on a small scale. A good example is the TravTek trial in Orlando, Florida [Taylor, 1991]. This combines a map of the Orlando area with location determination, route planning, real time road congestion data, and a limited yellow pages. Most of the data is contained on a CDROM. In consequence, the data integration and heterogeneity problems we tackle are avoided in this and most other ATIS's. We are creating a networked service instead, and a database that can be greatly expanded in scale, both in type and area of coverage.

6 References

Biliris, A., S. Dar, and N. Gehani. 1992. *Ode 1.1 User's Manual*.

Jagadish, H. V. and L. O'Gorman. Dec. 1989. "An Object Model for Image Recognition," *IEEE Computer*, vol. 22 , no. 12, pp. 33–42.

Jagadish, H. V. 1990. "On Indexing Line Segments," *Proceedings of the 16th International Conference on Very Large Data Bases*, Brisbane, Queensland, Australia, pp. 614–625.

Jagadish, H. V. May 1991. "A Retrieval Technique for Similar Shapes," *Proc. ACM-SIGMOD Int'l Conf. on the Management of Data*, Denver, CO.

Kim, W., A. C. Kim, and A. Dale. 1989. "Indexing Techniques for Object-Oriented Databases," *Object Oriented Concepts, Databases, and Applications*, W. Kim and F. H. Lochovsky (eds.), Addison Wesley, pp. 371–394.

Morehouse, S. March 1985. "ARC/INFO: A Geo-Relational Model for Spatial Information," *Proc. of AUTO-CARTO 7*, pp. 388–397.

Potmesil, M. 1993. Private Communication.

Story, G. and H. V. Jagadish. October 11, 1991. *A Data Model for Multimedia Documents,* Bell Labs TM 11256-911011-13TM.

Taylor, K. B. 1991. “TravTek - Information and Services Center,” *Proceedings of the IEEE Vehicle Navigation & Information Systems Conference*, Dearborn, MI, pp. 763–774.

Vijlbrief, T. and P. Oosterom. August 3-7, 1992. “The GEO++ System: An Extensible GIS,” *Proceedings of the 5th International Symposium on Spatial Data Handling*, P. Bresnahan, E. Corwin, and D. Cowen (eds.), University of South Carolina, Charleston, SC, pp. 40–51.

White, M. 1980. “A Survey of the Mathematics of Maps,” *Proc. of AUTO-CARTO IV*.

Whyatt, J. D. and P. R. Wade. 1988. “The Douglas-Peucker Line Simplification Algorithm,” *Bulletin of the Society of University Cartographers*, vol. 22, no. 1, pp. 17–27.

High Performance R-trees

Christos Faloutsos[*] *Ibrahim Kamel*

Department of Computer Science
University of Maryland at College Park

1 Introduction

One of the requirements for the database management systems (DBMSs) of the near future is the ability to handle spatial data [12]. Spatial data arise in many applications, including cartography, Computer-Aided Design (CAD) [7], computer vision and robotics, traditional databases (a record with k attributes corresponds to a point in a k-d space), temporal databases, medical image databases [1], scientific databases [3]. In the above applications, one of the most typical queries is the *range query*: Given a rectangle, retrieve all the elements that intersect with it.

Spatial access methods have attracted a huge amount of interest (see, eg. [10] for a recent survey). One of the most promising methods is the R-tree [7] and its variants, with the R^*-tree [2] showing the best performance. In R-trees, spatial objects are represented by their minimum bounding rectangles (MBR). The reader is assumed to be familiar with the R-tree structure.

We discuss two projects: (a) how to build better R-trees in a centralized environment and (b) how to decluster an above R-tree in a parallel setting. Section 2 examines the centralized case, and it describes how to exploit the good clustering properties of the Hilbert curve. The idea is to impose an ordering on the data items, so that they can be grouped into tight R-tree nodes. Section 3 shows how to exploit parallelism to improve the search performance of *any* R-tree structure. It proposes a new criterion, the 'proximity measure', to decide whether two rectangles are too close to each other to reside on the same unit. Section 4 lists the conclusions and future research directions.

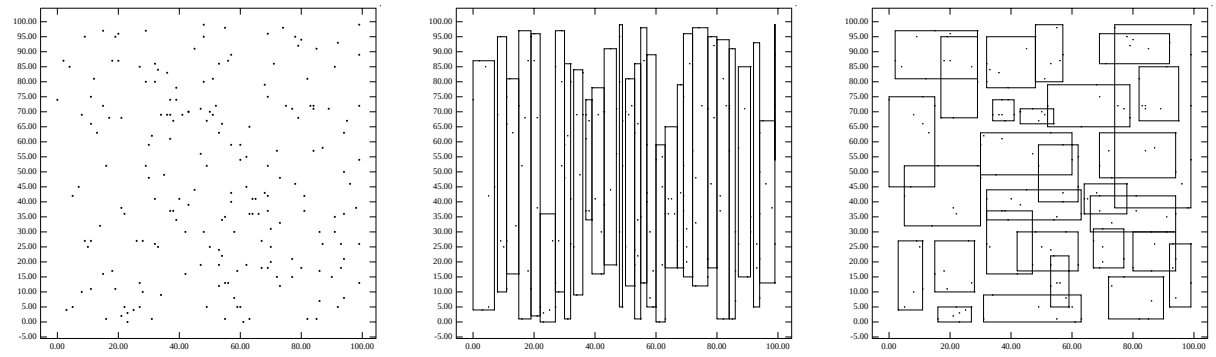
2 Centralized case: Hilbert-Rtree

For clarity of presentation, we temporarily assume that the data are points, and that the database is static. In such an environment, the only packing method for R-trees is the one proposed by Roussopoulos and Leifker [9]. They build a packed R-tree by sorting the data on the x or y coordinate of one of the corners of the rectangles. Once the rectangles have been sorted, the next step is to scan the list, assigning successive rectangles to the same R-tree leaf node, until this node is full; then, a new leaf node is created and the scanning of the sorted list continues. In our implementation of the method, we have chosen to sort on low-x; since our experiments showed no performance difference among sorting on low-x, high-x etc. Thus, we shall refer to this method as the *lowx* packed R-tree.

Although it performs very well for point queries on point data, its performance degrades for larger queries. Figures 1(a)-(b) highlight the problem. Figure 1(b) shows the leaf nodes of the R-tree that the *lowx* packing

[*]This research was partially sponsored by the Institute for Systems Research (ISR) and by the National Science Foundation under the grant IRI-8958546 (PYI), with matching funds from EMPRESS Software Inc. and Thinking Machines Inc..

method will create for the 200 random points of Figure 1(a). Neglecting the y-coordinate, the *lowx* packed R-tree tends to create elongated nodes, containing points that are not necessarily close to each other. To achieve



(a) (b) (c)

Figure 1: *(a) 200 points uniformly distributed; (b) grouped by the 'lowx packed R-tree' algorithm; (c) grouped by the proposed Hilbert-curve method.*

a better ordering, we propose to use space filling curves (or fractals), and specifically the Hilbert curve, since it is known for its superior clustering properties [4]. Figure 2 shows the Hilbert curves of order 1,2 and 3 (ie., for 2x2, 4x4 and 8x8 grids respectively). The number by each grid point is its 'Hilbert value'. Thus, the proposed packing method is similar to the *lowx* packed R-tree, with the major difference that the data points are sorted on their Hilbert values. Figure 1(c) shows the the MBRs of the resulting R-tree leaf nodes; notice that they tend to be small *square*-like rectangles. This indicates that the nodes will likely have small area and small perimeter, both of which properties will lead to faster response times (see Figure 3))

As we discuss in technical reports, the idea can be extended to handle rectangles (in addition to points) [6] as well as insertions and deletions [5], while still maintaining its search performance. The resulting method will be referred to as *Hilbert R-tree*.

Performance experiments in the above two technical reports confirm the intuitive arguments about the superiority of the Hilbert R-tree. Figure 3 gives a typical sample of these results, plotting the response time (disk accesses) as a function of the area of the query, on a file of real data, a 'TIGER' file, from the U.S. Bureau of Census, containing 39717 line segments (the roads of Montgomery county in Maryland). Figure 3(a) examines the static case, where the whole database is known in advance, and no modifications are allowed. It compares the proposed method (Hilbert R-tree) against the low-x, the R^*-tree and the R-tree with quadratic split. The proposed method achieves up to 36% improvement over the next best method (R^*-tree) and up to 58% improvement over the *lowx* packed R-tree. Figure 3(b) gives the results for the dynamic case (ie., insertions and deletions are permitted). Again, Hilbert R-tree is the clear winner, outperforming by up to 28% the next best competitor (again, the R^*-tree).

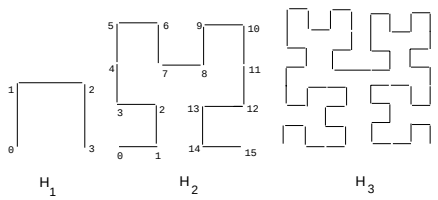
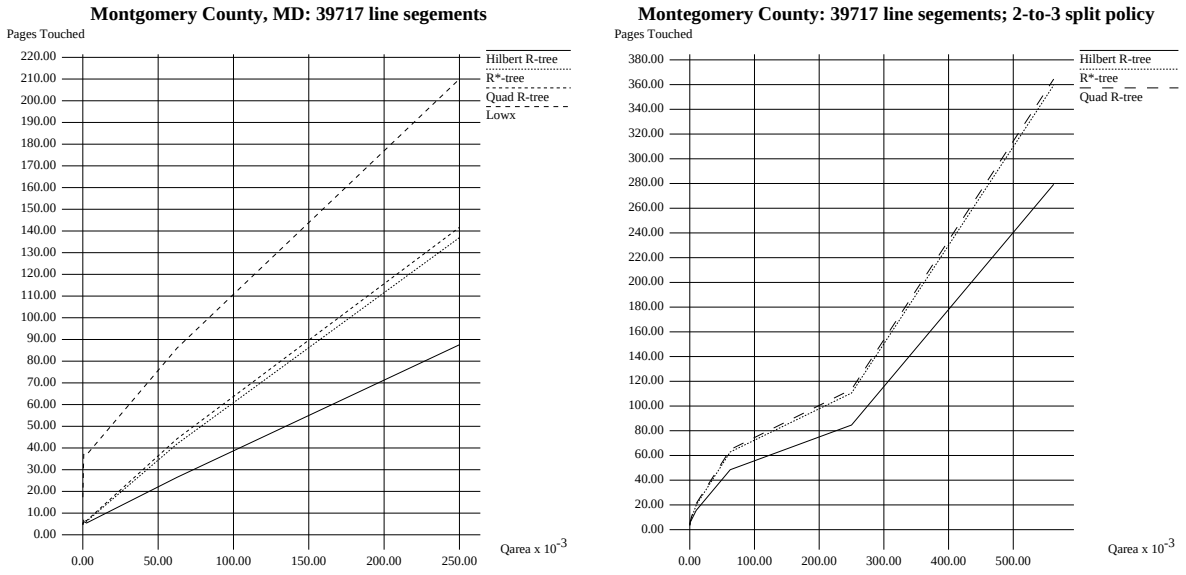


Figure 2: Hilbert Curves of order 1, 2 and 3.



(a) STATIC (b) DYNAMIC

Figure 3: *response time vs. query area - real data (MGCounty dataset)*

3 Parallel case: MX-R-tree

The goal of this project is to decluster the R-tree structure on a multi-disk architecture, to improve the performance for range queries. Notice that the declustering method is *orthogonal* to the choice of the specific R-tree variant; *any* of the known R-tree variants can be used (including the freshly introduced Hilbert R-tree).

We have examined a multi-disk architecture, consisting of one processor with several disks attached to it. There are two reasons for our choice: (a) Several of the target applications involve huge amounts of data, which do not fit in one disk. For example, NASA expects 1 Terabyte ($=10^{12}$) of data per day; the TIGER database of the U.S. Bureau of Census is 19 Gigabytes. (b) a multi-disk architecture can be used as a building block for a multi-processor share-nothing architecture, which is the topic of future research.

In the multi-disk environment, our goal is to maximize the throughput, which, as noted by Seeger and Larson [11], translates into the following two requirements: (a) *'minimum load':* Queries should touch as few nodes as possible, imposing a light load on the I/O sub-system. As a corollary, small queries (eg., point queries) should activate as few disks as possible. (b) *'uniform spread':* Nodes that qualify under the same query, should be distributed over the disks as uniformly as possible. As a corollary, queries that retrieve a lot of data should activate as many disks as possible.

We strive for a way to distribute an R-tree over d disks, to meet both of the above requirements. There are three major approaches to do that: (a) d independent R-trees (one R-tree per disk, with no pointers across disks); (b) Disk striping (or 'super-nodes', or 'super-pages' - one global R-tree, with each node consisting of d disk pages, striped across the d disks); and (c) the 'Multiplexed' R-tree, or *MX-R-tree* for short: one global R-tree, with single-page nodes, but with pointers across disks (see Figure 4).

As discussed in [8], the most promising approach is the third one. By its construction, the MX-R-tree fulfills the 'minimum load' requirement (as opposed, eg., to the super-node approach, which activates all d disks, even for the tiniest query). To meet the 'uniform spread' requirement, we need a good declustering method, that is, a heuristic to assign nodes to disks. The performance measure for these heuristics is the response time, that is, the completion time of the latest disk; the CPU costs are negligible and are ignored.

The only case where we need to worry about declustering is when a new R-tree node is created, that is, when

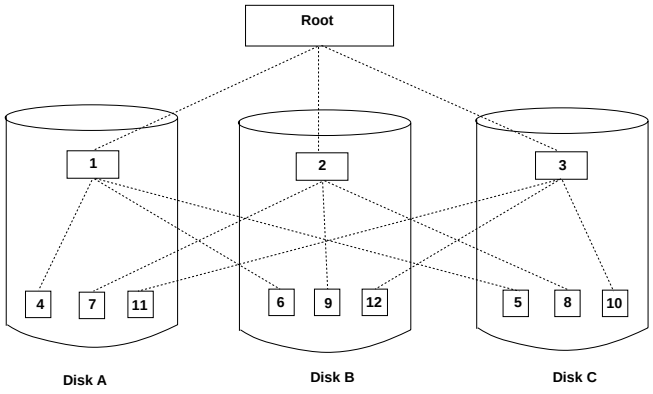


Figure 4: *R-tree stored on three disks.*

a node overflows and has to be split into two. One of these two new nodes, say, N_0, has to be assigned to another disk. To keep the insertion time short, we consider only the *sibling* nodes $N_1, \ldots, N_k$, that is, the nodes that have the same father N_{father} with N_0 (see Figure 5 for an illustration). Thus, the problem can be informally stated as follows: **Given** a node (= rectangle) N_0, a set of nodes $N_1, \ldots, N_k$ and the assignment of nodes to disks **Assign** N_0 to a disk, to optimize the response time on range queries.

We have experimented with several heuristics, such as: (a) Round-Robin ('RR'), which strives for 'data balance', that is, for equal number of nodes per disk. (b) Minimum Area ('MA'), which strives for 'area balance', that is, each disk should roughly cover the same area in native space. For example, in Figure 5, MA would assign N_0 to disk A, because the light gray rectangles N_1, N_3,N_4 and N_6 of disk A have the smallest sum of area. (c)

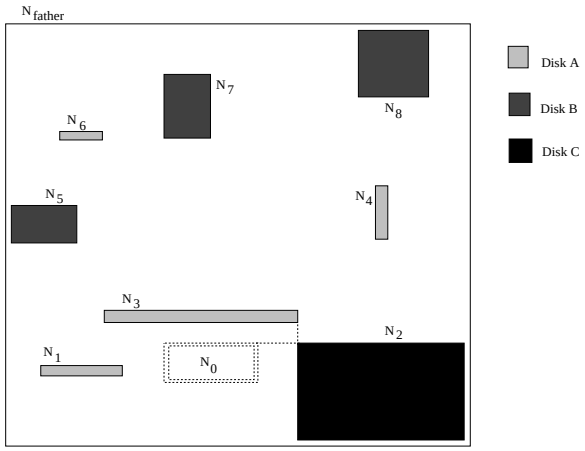


Figure 5: *Node N_0 is to be assigned to one of the three disks.*

Minimum Intersection ('MI'), which tries to minimize the overlap of nodes that belong to the same disk.

The proposed heuristic is the Proximity Index ('PI'), which assigns the new node N_0 to the disk with the 'least similar' nodes with respect to N_0 (ie., least likely to qualify in the same range query with N_0). For the setting of Figure 5, PI will assign N_0 to disk B because it contains the most remote rectangles. Intuitively, disk B is the best choice for N_0, in our effort to satisfy the 'uniform spread' criterion. The formulas to compute the proximity index of a node (rectangle) N_0 to a disk that contains a *set* of rectangles N_1, ..., N_k is based on the *proximity measure*: This measure compares two rectangles and assesses the probability that they will be retrieved by the same query. The *proximity index* of a new node N_0 and a disk D (which contains the sibling nodes $N_1, \ldots, N_k$) is the proximity of the most 'proximal' node to N_0.

The exact formulas for the proximity measure and the proximity index are in [8]. For brevity, we present only the formulas for 1-d case: If R and S are two line segments (1-d rectangles), then

$$proximity(R,S) = \begin{cases} 1/3 \times (1 + 2 \times \delta) & \text{if } R \text{ and } S \text{ intersect} \\ 1/3 \times (1 - \Delta)^2 & \text{if } R \text{ and } S \text{ are disjoint} \end{cases} \quad (1)$$

where δ is the length of the common segment, if R and S intersect, and Δ is the distance between them, if they are disjoint.

The same paper reports simulation results; we present some of those graphs next. Figure 6 compares the PI heuristic over the rest (RR, MA, MI), for the MX-R-tree. It shows the response time as a function of the side q_s of the (square) queries. Notice that PI and MI, the two heuristics that take the spatial relationships into account,

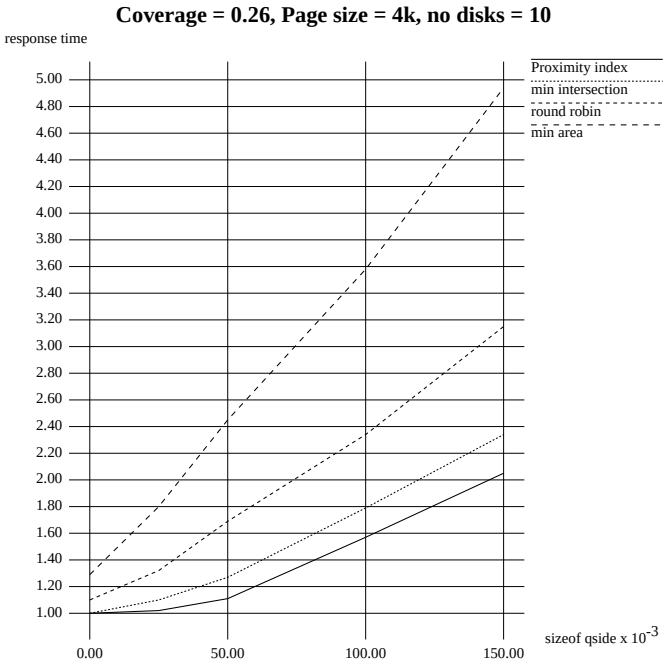


Figure 6: *Comparison among all heuristics (PI,MI,RR and MA)- response time vs query size*

perform the best. Round Robin is the next best, and the Minimum Area heuristic is last.

Additional experiments in [8] compare the MX-R-tree (with PI)

against the super-node (= disk striping) approach, showing that the MX-R-tree typically achieves better response times, while it always achieves lower load, as expected. Experiments in the same paper show that the MX-R-tree scales up well.

4 Conclusions - Future research

In this paper we described two projects, which they both target high-performance R-tree designs. For the centralized case, we discussed a superior R-tree variant, which uses the Hilbert curve, and outperforms all the previous R-tree methods. The major idea is to introduce a 'good' ordering among data rectangles.

In a parallel environment, with one CPU and multiple disks, we proposed the 'proximity index' (PI) criterion, to help decluster the nodes of an R-tree. Specifically, this criterion tries to store a new node on that one disk that contains nodes as dissimilar to the new node as possible.

Future research could focus on the following topics: (a) the use of the proximity concept to aid the parallelization of other spatial file structures; (b) declustering schemes for multi-processor, share-nothing architectures; and (c) analysis of the Hilbert R-trees, providing analytical formulas that predict the response time as a function of the characteristics of the data rectangles (count, data density etc).

References

[1] Manish Arya, William Cody, Christos Faloutsos, Joel Richardson, and Arthur Toga. Qbism: a prototype 3-d medical image database system. *IEEE Data Engineering Bulletin*, March 1993.

[2] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The r*-tree: an efficient and robust access method for points and rectangles. *ACM SIGMOD*, pages 322–331, May 1990.

[3] Mathematical Committee on Physical and NSF Engineering Sciences. *Grand Challenges: High Performance Computing and Communications*. National Science Foundation, 1992. The FY 1992 U.S. Research and Development Program.

[4] C. Faloutsos and S. Roseman. Fractals for secondary key retrieval. *Eighth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 247–252, March 1989. also available as UMIACS-TR-89-47 and CS-TR-2242.

[5] Christos Faloutsos and Ibrahim Kamel. Hilbert R-tree: an improved R-tree using fractals. Systems Research Center (SRC) TR-93-19, Univ. of Maryland, College Park, 1993.

[6] Christos Faloutsos and Ibrahim Kamel. Packed R-trees using fractals. Systems Research Center (SRC) TR-93-1, Univ. of Maryland, College Park, 1993.

[7] A. Guttman. R-trees: a dynamic index structure for spatial searching. *Proc. ACM SIGMOD*, pages 47–57, June 1984.

[8] Ibrahim Kamel and Christos Faloutsos. Parallel R-trees. *Proc. of ACM SIGMOD Conf.*, pages 195–204, June 1992. Also available as Tech. Report UMIACS TR 92-1, CS-TR-2820.

[9] N. Roussopoulos and D. Leifker. Direct spatial search on pictorial databases using packed R-trees. *Proc. ACM SIGMOD*, May 1985.

[10] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, 1989.

[11] Bernhard Seeger and Per-Ake Larson. Multi-disk b-trees. *Proc. ACM SIGMOD*, pages 138–147, May 1991.

[12] Avi Silberschatz, Michael Stonebraker, and Jeff Ullman. Database systems: Achievements and opportunities. *Comm. of ACM (CACM)*, 34(10):110–120, October 1991.

Using the Holey Brick Tree for Spatial Data in General Purpose DBMSs*

Georgios Evangelidis Betty Salzberg

College of Computer Science
Northeastern University
Boston, MA 02115-5096

1 Introduction

There is leverage to be gained by bringing spatial data within the purview of general purpose database systems. A spatial access method embedded in a general purpose DBMS would have several advantages [Lom91]:

1. Spatial data could be integrated with other data. Spatial objects are likely to have attributes other than the location of their bounding boxes. These other attributes can be queried using traditional methods.
2. Multiple users (some making updates) would be supported by concurrency algorithms already in place.
3. In case of a system failure, the restart process would be able to recover the database to a consistent state.
4. Robust system utilities for loading data, analyzing performance, producing reports and so forth could be applied.

To this end, we have been modifying the hB-tree (or holey Brick tree) [LS90], an efficient multi-attribute search structure, for use with the concurrency and recovery systems available in general purpose DBMSs. This will make it more likely that vendors of DBMSs will be able to provide support for spatial data.

In section 2, we review the hB-tree. In section 3, we show why structures which cluster points in space as the hB-tree does will also be efficient for clustering k-dimensional spatial objects when their bounding box coordinates are entered as points in $2k$-dimensional space. We give here performance results which demonstrate that the hB-tree is especially well suited for such spatial data, since the space used for the index is not sensitive to the number of dimensions. In section 4, we indicate how the concurrency and recovery algorithm of [LS92] can be applied to a modified hB-tree.

2 The hB-tree

The hB-tree is a multi-attribute index structure. Its inter node search and growth processes are precisely analogous to the corresponding processes in B-trees. Its nodes represent bricks (i.e., multi-dimensional rectangles), or "holey" bricks, that is, bricks from which smaller bricks have been removed.

It consists of **index** and **data** nodes. Index nodes are responsible for multi-dimensional subspaces. They contain a **kd-tree** [Ben79] which is used to organize information about lower levels of the hB-tree and extracted

*This work was partially supported by NSF grant IRI-91-02821.

index-level regions. Data nodes contain the actual data, which are points in multi-dimensional space. Since the space a data node describes may be holey, data nodes may contain kd-trees, as well, that enable the data nodes to describe their own "inner" boundaries and the extracted data-level regions.

A kd-tree is a binary tree that is capable of describing a region (possibly holey) of the multi-dimensional space of attributes. The leaf nodes of a kd-tree may: (a) refer to other hB-tree nodes of the next lower level, (b) be markers, called **external markers**, indicating that a subtree is missing (this happens when a node is split and part of the attribute space it was describing is now described by another sibling node), or (c) refer to a collection of data records that are kept in a **record list** (in data nodes, only).

Figure 1 contains an example hB-tree. We have a two-dimensional attribute space. The root of the hB-tree is the index node I, that describes the whole space. Its kd-tree describes the next level of the hB-tree, which consists of two data nodes. All records with attributes $x \in (x_0, +\infty)$ and $y \in (y_0, +\infty)$ are located in data node B, and the rest of the records are located in data node A.

A's kd-tree shows that A, which initially was responsible for the whole attribute space, was split and part of its records moved to node B (that is what the external marker denotes).

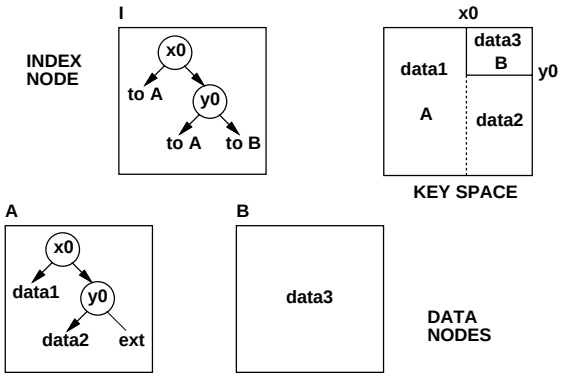


Figure 1: An example hB-tree.

2.1 Searching

Searching for point data using the hB-tree is straightforward. We start the search at the hB-tree root. The root is searched by traversing its kd-tree. Every kd-tree node has information which includes an attribute and its value. By comparing this value to the value of the corresponding attribute of the search point, we can decide on which of the two children of the kd-tree node we should visit next.

This process will lead us to lower levels of the hB-tree, and eventually to a data node. That data node contains the part of the attribute space where the search point belongs. Finally, the points of the node are searched with the help of the kd-tree of the node.

2.2 Node Splitting and Index Term Posting

The idea behind node splitting is the same as in B-trees. When a node becomes full it has to be split. Its kd-tree is used to find a subtree whose size is between one and two thirds of the contents of the node. A new hB-tree node is allocated and the extracted contents are moved to it. An external marker is included in the original node indicating that part of its contents have been extracted.

The kd-tree nodes in the path from the kd-tree root of the node to the extracted subtree which describe the extracted region and have not been posted during another splitting are posted to the parent hB-tree node. Posting of index terms may trigger additional node splits at higher levels.

3 Using the hB-tree for Spatial Objects

The hB-tree can be used for spatial objects as can any other multiattribute point index. The spatial objects are represented in the index by the coordinates of their bounding boxes.

We show first that any multiattribute index (such as the hB-tree) which clusters data points which are nearby in space, will, using boundary coordinates, cluster together records of nearby spatial objects. Since the number

of coordinates used to describe a bounding box is twice the number of dimensions of the space, indexes whose size does not depend on the number of coordinates of the data points will be superior. We show that the hB-tree disk space utilization is insensitive to the dimension of the space.

3.1 Mapping Spatial Objects to Points

Say we are using $2k$ coordinates to describe a brick in k-dimensional space. If two points in the $2k$-dimensional space have similar values in all coordinates, (a) the objects are similar in size, (b) if the objects are small, they are close in space, and (c) if the objects are large, they will overlap. Thus, any method which clusters nearby points will group records of large objects together in pages with other overlapping large objects. It will also group small objects together with nearby small objects in pages.

This clustering is efficient for typical spatial queries. Large objects are likely to be the answer to many queries. Having them clustered in disk pages will increase the locality of reference. Having the small objects clustered together will decrease the number of pages which must be accessed for a particular query [Lom91].

To illustrate these points, we look at one-dimensional spatial objects, which are line segments with a begin value and an end value. We map these objects to points in two-dimensional space using the x-coordinate for the begin value of the line segment and the y-coordinate for the end value. Since the begin value is always less than or equal to the end value, all points will lie on or above the line $x = y$. Points representing small line segments will be near the diagonal line $x = y$ and points representing large line segments will be far from the line $x = y$ (figure 2).

Common spatial queries such as inclusion or intersection are represented by rectangular bricks in the transformed ($2k$-dimensional) space. For example, as illustrated in figure 2, all line segments containing the point "6" would be in the area where the begin value x is less than or equal to 6 and where the end value y is greater than or equal to 6.

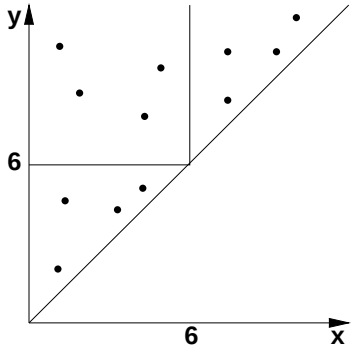


Figure 2: Mapping of line segments to points. Points close to the line $x = y$ will represent small line segments.

3.2 Performance of the hB-tree in High Dimensions

The main objection to using multiattribute point-based methods for spatial objects is that the number of dimensions needed to represent the objects doubles, making the index too large. But the hB-tree is especially insensitive to increases in dimension.

A kd-tree node always stores the value of exactly one attribute. Thus, the size of a kd-tree node (and, consequently, the size of the kd-trees that reside in the hB-tree nodes) does not depend on the number of indexing attributes.

But, in addition to a kd-tree, every hB-tree node stores its own boundaries (i.e., low and high values for all attributes that describe the space the node is responsible for). These are $2k$ attribute values for a k-dimensional hB-tree. An increase on the number of dimensions does increase the space required to store a node's boundaries. The additional amount of space required as the dimensions increase becomes a non-factor for large enough page sizes. Figure 3 illustrates this fact. With a page size of 2K bytes and larger, there is almost no effect on the size of the hB-tree and the node space utilization as the dimensions increase. (Page sizes larger than 2K bytes are not shown.)

This is in contrast, for example, with the R-tree [Gut84], where index entries are bounding coordinates of objects plus a pointer. Thus, in the R-Tree (and its variants) the size of the index is proportional to the dimension of the space. The grid file [NHS84] is even worse since it is a multidimensional array. Thus doubling the number

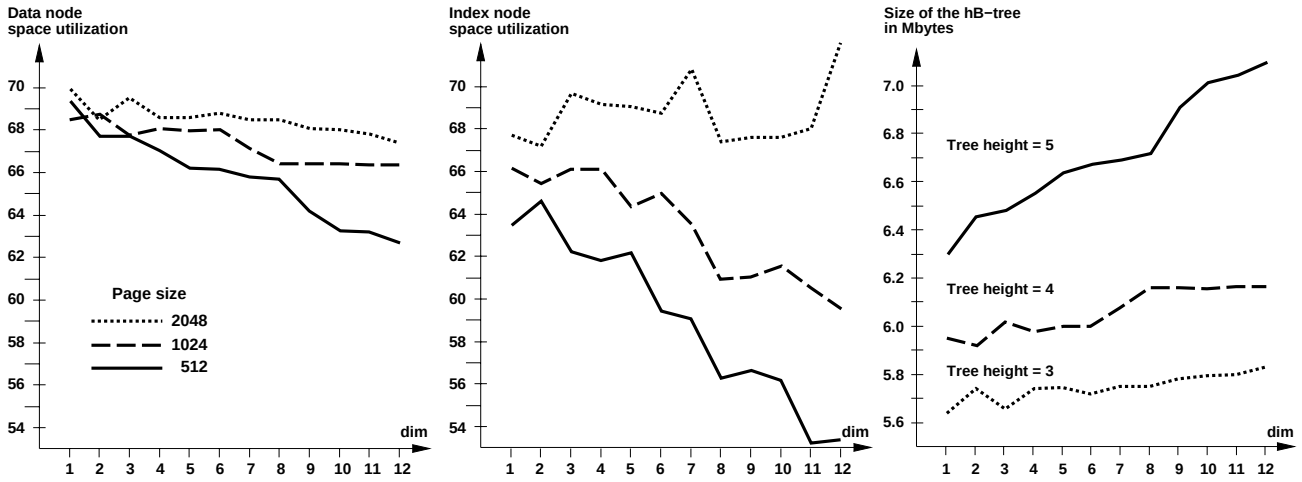


Figure 3: Index and data hB-tree node space utilization and size of the hB-tree in terms of height and Mbytes under different page sizes and dimensions. For page sizes greater than 2K bytes the hB-tree is insensitive to the number of dimensions. In all cases the same 150,000 24-byte records were inserted.

of dimensions squares the size of the index. In the worst case it is an $O(n^k)$ size index, where n is the number of points and k is the dimension of the space. Z-ordering, [OM84] on the other hand, like the hB-tree is insensitive to dimension.

4 Dealing with Concurrency and Recovery

New access methods which are advantageous for spatial data cannot be integrated into general purpose DBMSs unless they use the existing concurrency and recovery methods provided by the systems. The hB-tree has been properly modified so that the efficient algorithm for concurrency and recovery of [LS92] is applicable on it.

The most important modification is the replacement of external markers by actual pointers to the extracted nodes. The hB-tree becomes a subcase of the Π-tree [LS92], an abstract tree-structure which is a generalization of the B^{link}-tree [LY81]. We call the modified hB-tree an hB^{Π}-tree. Now, we can be lazy about posting index terms that describe a node split. In other words, two instances of the hB^{Π}-tree can be structurally different, because some index term postings have not been performed, but semantically equivalent.

4.1 Splitting and Posting

As with simple B-trees, when an insertion causes an hB^{Π}-tree node to overflow, that node is split, with part of the contents going to a new sibling node, and a new index term is posted to the parent. In the hB^{Π}-tree the node splitting phase and the index posting phase are performed by separate atomic actions, as following:

Node splitting phase: An updater detects an hB^{Π}-tree node that is full and cannot accommodate the update. This node, called the **container node**, is split and part of its contents are moved to a newly created node, called the **extracted node**. The node splitting phase concludes by storing a side pointer in the container node that points to the extracted node (figure 4b).

Index posting phase: An index term that describes the space that was extracted from the container is posted to a parent of the container. Since an hB^{Π}-tree node may have many parents, the index posting phase may consist of several separate **index posting atomic actions**. An index posting atomic action always posts to a single parent (figure 4c).

An index posting atomic action can be scheduled after: (a) **a node splitting phase is over**, in which case it involves the parent of the container node that lies in the current search path, or (b) **a side pointer traversal**, which is an indication that either the posting action that was scheduled after the splitting was not performed for some reason, or the container node is a multiparent node and the current search path is through a parent other than the one involved in the node splitting phase.

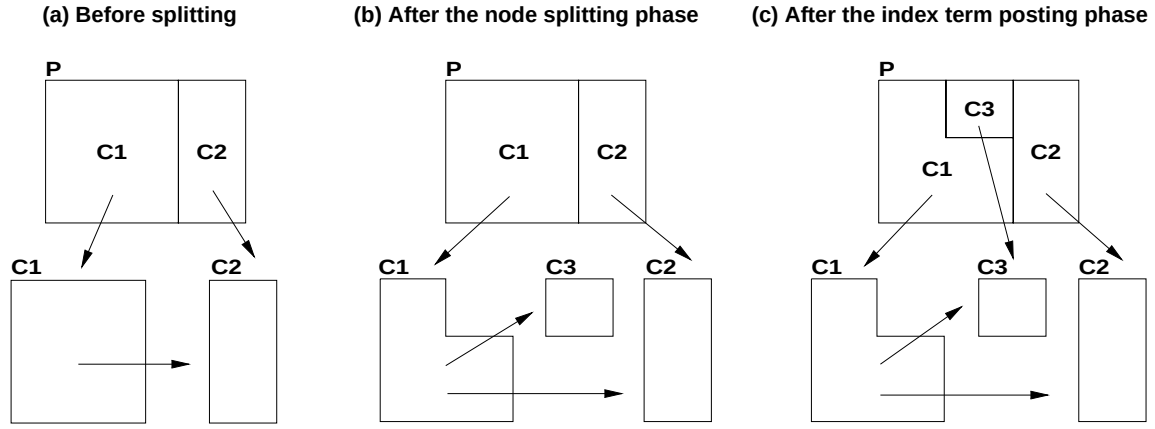


Figure 4: In the node splitting phase, node C1 is split, part of its contents move to the new node C3 and a side pointer to C3 is included in C1. In the index term posting phase an index term that describes the space of C3 is posted to its parent P.

In case an index posting atomic action can not take place for any reason, the hB$^{\Pi}$-tree is left in a consistent state. Searchers can always traverse or visit the extracted node by going through its container node and following the side pointer. System failure is not the only thing that prevents posting actions from completing. Depending on the algorithm used for performing the posting of index terms, a posting action may be dropped because the algorithm decides that it is not a good idea to perform the posting at that particular moment for performance reasons. Being too lazy in posting index terms may degrade the performance of the hB$^{\Pi}$-tree. An algorithm has to balance the tradeoff between an efficient and simple scheme for posting index terms and more expensive traversals of the hB$^{\Pi}$-tree due to a large number of side pointer traversals.

In the hB$^{\Pi}$-tree we have found that if we post more information (extra kd-tree nodes) than is actually needed, or if we restrict index node splits to certain places on their kd-tree, our algorithms become simpler. Such simplification could have negative consequences on performance since worst case guarantees of index term size and index node space utilization no longer hold as in the hB-tree. However, our preliminary experiments show that even with simple algorithms the performance is very good [ELS93].

5 Conclusions

The hB-tree is especially well-suited for spatial data. It clusters points using median attributes. It is insensitive to increases in dimension. Like any other point-clustering method, it can be expanded for use with bounding boxes of spatial objects. The resulting clustering will be effective for typical spatial queries such as nearest-neighbor and intersection searching. In addition, we are applying an efficient concurrency and recovery method to a modified hB-tree (called the hB$^{\Pi}$-tree) making possible its integration into a general purpose DBMS. This will enable users of spatial data to take advantage of the extensive and robust capabilities of these systems.

References

[Ben79] J. L. Bentley. Multidimensional binary search trees in database applications. *IEEE Transactions on Software Engineering*, SE-5(4):333–340, July 1979.

[ELS93] G. Evangelidis, D. Lomet, and B. Salzberg. Towards a concurrent and recoverable multiattribute indexing method. Technical Report in preparation, College of Computer Science, Northeastern University, 1993.

[Gut84] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of ACM/SIGMOD Annual Conference on Management of Data*, pages 47–57, Boston, MA, June 1984.

[Lom91] D. Lomet. Grow and Post Index Trees: role, techniques and future potential. 2nd Symposium on Large Spatial Databases (SSD91) (August, 1991) Zurich. In *Advances in Spatial Databases, Lecture Notes in Computer Science 525*, pages 183–206, Berlin, 1991. Springer-Verlag.

[LS90] D. Lomet and B. Salzberg. The hB-Tree: A multiattribute indexing method with good guaranteed performance. *ACM Transactions on Database Systems*, 15(4):625–658, December 1990.

[LS92] D. Lomet and B. Salzberg. Access method concurrency with recovery. In *Proceedings of ACM/SIGMOD Annual Conference on Management of Data*, pages 351–360, San Diego, CA, June 1992.

[LY81] P. Lehman and S. B. Yao. Efficient locking for concurrent operations on B-trees. *ACM Transactions on Database Systems*, 6(4):650–670, December 1981.

[NHS84] J. Nievergelt, H. Hinterberger, and K. C. Sevcik. The Grid File: An adaptable, symmetric, multikey file structure. *ACM Transactions on Database Systems*, 9(1):38–71, March 1984.

[OM84] J. A. Orenstein and T. Merrett. A class of data structures for associative searching. In *Proceedings of SIGART-SIGMOD 3rd Symposium on Principles of Database Systems*, pages 181–190, Waterloo, Canada, 1984.

Definitions of Line-Line Relations for Geographic Databases*

Max J. Egenhofer
National Center for Geographic Information and Analysis
and
Department of Surveying Engineering
Department of Computer Science
University of Maine
Orono, ME 04469-5711
max@mecan1.maine.edu

Abstract

Query languages for spatial databases need appropriate tools to inquire about spatial data and provide access to the relations among spatial objects. These spatial relations are more complex than conventional predicates comparing equality or order. Examples are such spatial predicates as "neighbor," "intersect," and "inside." A formal definition of spatial relations is necessary to define the semantics of an appropriate set of spatial predicates in query languages and to provide a basis for spatial query processing. We have extended a model, initially designed for binary topological relations between 2-dimensional objects, to treat 1-dimensional objects in $\mathbb{R}^2$ as well. The approach used is based upon algebraic topology and compares the interiors, boundaries, and exteriors of the lines. A total of 33 different topological relations between two simple lines has been identified formally, for which geometric interpretations are given.

1 Introduction

Queries in Geographic Information Systems (GISs) and image databases are often based on the relations among spatial objects. For example, in geographic applications typical spatial queries are, "Retrieve all roads that lead to interstate highway I-95" and "Find all electric power lines that run across a river." Current database query languages, such as SQL and Quel, do not sufficiently support such queries, because they provide only tools for comparing equality or order of such simple data types as integers or strings. The incorporation of spatial relations over geometric domains into a spatial query language has been identified as an essential extension beyond the power of traditional query languages [5, 17]. Some experimental spatial query languages support queries with one or the other spatial relation; however, the diversity, semantics, completeness, and terminology vary dramatically [4, 10]. With the advent of extensible database query languages such as SQL3, it will become increasingly important to have (consistent) models and formalizations of relations.

Previous investigations developed a formal model for *binary topological relations* for co-dimension 0 (i.e., if the dimension between the embedding space and the objects is 0). It applies to relations between regions (2-dimensional objects) embedded in $\mathbb{R}^2$ [7] or lines in $\mathbb{R}^1$ [16]. For these settings, we have promoted a comprehensive formalism [6], which generalizes to n-dimensional objects embedded in $\mathbb{R}^n$. Others have used this

*This research was partially funded by NSF grant No. IRI-9309230 and grants from Intergraph Corporation. Additional support from NSF for the NCGIA under No. SES-880917 is gratefully acknowledged.

model, for instance to formalize topological relations among spatial objects in 3-D [11, 15], studies of the use of spatial predicates in natural languages [14], or as a basis for the integration of topological constraints into spatial query languages [3, 13, 18]. Here, we extend the formalizations of topological relations to line-line relations in $\mathbb{R}^2$.

The remainder of this paper is organized as follows: Section 2 briefly summarizes the model used for topological relations with co-dimension 0 and demonstrates why it is necessary to extend it for objects with co-dimension 1, such as lines in $\mathbb{R}^2$. Section 3 introduces the 9-intersection as our model to formalize binary topological relations between lines in $\mathbb{R}^2$. Section 4 investigates which relations can be realized between two lines in $\mathbb{R}^2$. The conclusions in Section 5 discuss future research activities.

2 4-Intersection

Binary topological relations between two objects, A and B, are defined in terms of the four intersections of A's boundary (∂A) and interior (A°) with the boundary (∂B) and interior (B°) of B [6], called the *4-intersection* [6]. Topological invariants of these four intersections, i.e., properties that are preserved under topological transformations, are used to categorize topological relations. Examples of topological invariants applicable to the 4-intersection are the content (i.e., emptiness or non-emptiness) of a set, the dimension, and the number of separations [9]. The content invariant is the most general criterion as any other invariants can be considered refinements of non-empty intersections. By considering the values empty ($\emptyset$) and non-empty ($\neg\emptyset$) for the four intersections, one can distinguish sixteen binary topological relations, nine of which can be realized for 2-dimensional objects (including objects with holes), called *regions*, if the objects are embedded in $\mathbb{R}^2$ [7], and eight between two *lines* in $\mathbb{R}^1$ (Fig. 1) (which correspond to Allen's interval relations [2] if the order of the 1-dimensional space is disregarded) . The difference is due to the fact that regions may have connected boundaries, while lines have disconnected boundaries.

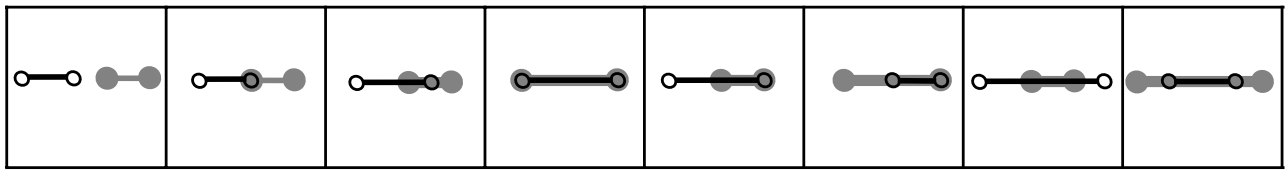


Figure 1: Examples of the eight topological relations between two lines in $\mathbb{R}^1$.

The same relations also exist when the objects are mapped into a higher-dimensional space; however, due to the greater degree of freedom, the objects may take configurations that are not represented by one of the relations between objects with co-dimension 0. For example, if two lines "cross," they have non-empty interior-interior

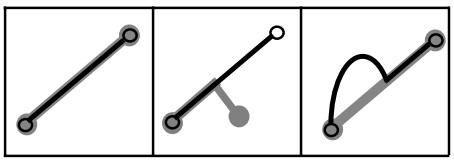


Figure 2: Examples of topological relations between two lines in $\mathbb{R}^2$ that have the same 4-intersection ($A^\circ \cap B^\circ = \neg\emptyset$; $A^\circ \cap \partial B = \emptyset$; $\partial A \cap B^\circ = \emptyset$; and $\partial A \cap \partial B = \neg\emptyset$).

intersections, while the other three intersections are empty. Such a 4-intersection could not be realized for two lines in $\mathbb{R}^1$. On the other hand, some 4-intersections may have ambiguous geometric interpretations (Fig. 2).

¿From a practical point of view, there are certainly situations in which one would like to distinguish between them when querying a geographic database. These observations motivated the development of an extended model to account also for relations between n-dimensional objects that are embedded in an m-dimensional space, $m > n$.

3 9-Intersection for Binary Topological Relations between Lines

The definition of a line is based on 1-cells, i.e., the connections between two geometrically independent nodes. A line is a sequence of $1 \ldots n$ connected 1-cells such that they neither cross themselves nor form a cycle. Nodes at which exactly one 1-cell ends will be referred to as the *boundary* of the line. Nodes that are an endpoint of more than one 1-cell are *interior nodes*. The *interior* of a line is the union of all interior nodes and all connections between the nodes. The *closure* of a line is the union of its interior and boundary. Finally, the *exterior* is the difference between the embedding space and the closure of the lines. We will call a sequence of 1-cells a *simple line* if it has exactly two boundary nodes. Lines that would have less than two boundary nodes would include cycles, which are excluded by definition.

The 4-intersection is extended by considering the location of each interior and boundary with respect to the other object's exterior; therefore, the binary topological relation R between two lines, A and B, in $\mathbb{R}^2$ is based upon the comparison of A's interior (A°), boundary (∂A), and exterior (A^-) with B's interior (B°), boundary (∂B), and exterior (B^-). These six object parts can be combined such that they form nine fundamental descriptions of a topological relation between two lines and be concisely represented by a 3×3-matrix, called the *9-intersection*.

$$R(A,B) \quad = \quad \begin{pmatrix} A^\circ \cap B^\circ & A^\circ \cap \partial B & A^\circ \cap B^- \\ \partial A \cap B^\circ & \partial A \cap \partial B & \partial A \cap B^- \\ A^- \cap B^\circ & A^- \cap \partial B & A^- \cap B^- \end{pmatrix}$$

Each intersection will be characterized by a value *empty* ($\emptyset$) or *non-empty* ($\neg\emptyset$), which allows one to distinguish $2^9 = 512$ different configurations. Only a small subset of them can be realized between two lines in $\mathbb{R}^2$.

4 Existing 9-Intersections Between Lines in $\mathbb{R}^2$

In order to identify which of the 512 different 9-intersections may be realized between two lines in $\mathbb{R}^2$, we formalize a set of properties as conditions for binary topological line-line relations, that must hold between the parts of any two lines. These properties can be expressed as consistency constraints in terms of the 9-intersection [8], such that by successively eliminating from the set of 512 relations the relations that would violate a consistency constraint, one retains the candidates for those 9-intersections that can be realized for the particular spatial data model. The existence of these relations is then proven by finding geometric interpretations for the corresponding 9-intersections.

Condition 1: The exteriors of two lines always intersect with each other.

Condition 2: A's boundary intersects with at least one part (interior, boundary, or exterior) of B, and vice-versa.

Condition 3: Each boundary of a simple line intersects with at most two parts of another line.

Condition 4: If A's boundary is a subset of B's boundary, then the two boundaries coincide, and vice-versa.

Condition 5: If A's interior does not intersect with B's exterior then the two interiors must intersect as well, and vice-versa.

Condition 6: If A's interior does not intersect with B's exterior then A's boundary must not intersect with B's exterior, and vice-versa.

Condition 7: If A's interior does not intersect with B's exterior then A's interior must not intersect with B's boundary, and vice-versa.

Condition 8: If A's closure is a subset of B's interior then either A's exterior intersects with both B's boundary and B's interior, or not at all, and vice-versa.

Based on these conditions, we find 33 relations between two simple lines, 13 of which are symmetric (examples of geometric interpretations are shown in the top two rows of Fig. 3) and the remaining ones form 10 pairs of converse relations (bottom two rows of Fig. 3 show one example of each pair of the converse relations).

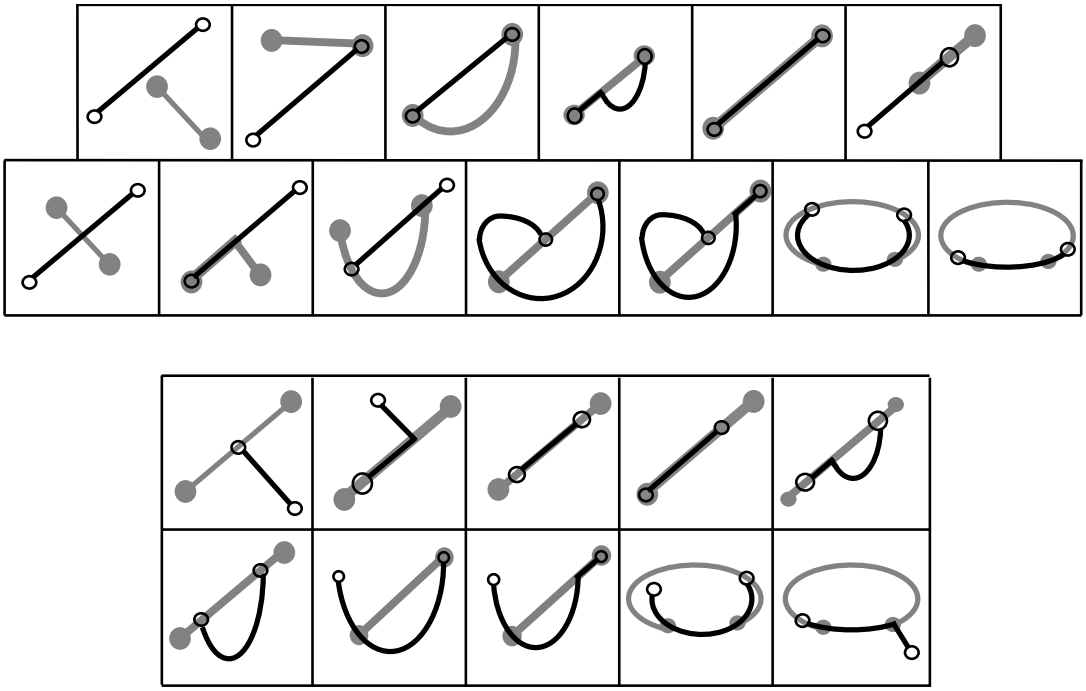


Figure 3: Geometric interpretations of the topological relations between two simple lines in $\mathbb{R}^2$.

5 Conclusions

A formalism for the definition of binary topological relations between linear spatial objects embedded into $\mathbb{R}^2$ has been presented. Binary topological relations are described by the 9-intersection, i.e., the set intersections between the interiors, boundaries, and exteriors of the two lines. The criterion for distinguishing different topological relations is the *content* of the 9-intersections, i.e., whether the intersections are empty or non-empty. Using these criteria to classify topological relations, it was shown that in $\mathbb{R}^2$ there are only 33 line-line relations that can be realized. This set of relations provides a solid framework to process spatial queries. While the content invariant provided a very generic criterion for classifying line-line relations, there are a number of other topological invariants that reveal more detailed differences about line relations. Examples of these invariants are the number of non-empty interior-interior intersections, their dimension, their type (crossing or touching), and the order in which these invariants are encountered [3, 9, 12]. Each of them increases the number of spatial relations a user can choose from when querying a geographic database.

The results of this paper have some serious implications for the design of spatial query languages. Obviously, there is a large number of different relations that users might want to distinguish. Although one may be tempted to group them into categories of conceptually similar relations [3], human subject tests with line-region relations have indicated that some users actually distinguish the entire array of different relations offered by the 9-intersection model [14]. The diversity of spatial configurations that may be distinguished and sometimes necessary for users to make decisions, may be greater than what can be easily described in a traditional database query language. For example, natural (English) language offers only a small, limited amount of spatial predicates to describe spatial configurations [19]. Without comprehensive tests of how humans judge spatial relations in different situations in which the context or the semantics of the objects involved change, any decisions about the use of vocabulary in spatial query languages will be speculative.

As a consequence for spatial query language design, detailed spatial relations may need to be described by other means than predicates, e.g., graphically as a sketch, as annotated drawing, or by selecting combinations of legal and illegal configurations from a set of given prototypes. Graphical renderings appear to be very natural in interacting with spatial data, however, there are some major problem one faces with specifying spatial constraints as sketches. For example, complex Boolean combinations of spatial constraints—particularly disjunctions and negation—are very difficult to draw. Likewise, any graphical rendering contains inevitably more spatial constraints than what is expressed by a single spatial predicate. For example, a sketch of a topological relation that "A is disjoint from B" carries additional information about the directions between the objects, their shapes, relative sizes and distances. Query languages that incorporate an *interview process* during which the user is involved in resolving overspecified or underspecified constraints, may be promising approaches to overcome the limitations of traditional query languages based on the question-answer paradigm.

References

[1] P. Alexandroff, *Elementary Concepts of Topology*. New York, NY: Dover Publications, Inc., 1961.

[2] J. Allen, "Maintaining knowledge about temporal intervals," *Communications of the ACM*, vol. 26, no. 11, pp. 832–843, Nov. 1983.

[3] E. Clementini, P. Di Felice, and P. van Oosterom, "A small set of formal topological relationships suitable for end-user interaction," in *Third International Symposium on Large Spatial Databases, SSD '93*, (Singapore), pp. 277–295, Lecture Notes in Computer Science, vol. 692, New York, NY: Springer-Verlag, June 1993.

[4] M. Egenhofer, *Spatial Query Languages*. PhD thesis, University of Maine, Orono, ME, 1989.

[5] M. Egenhofer and A. Frank, "Towards a spatial query language: user interface considerations," in *14th International Conference on Very Large Data Bases*, (D. DeWitt and F. Bancilhon, eds.), (Los Angeles, CA), pp. 124–133, Aug. 1988.

[6] M. Egenhofer and R. Franzosa, "Point-set topological spatial relations," *International Journal of Geographical Information Systems*, vol. 5, no. 2, pp. 161–174, 1991.

[7] M. Egenhofer and J. Herring, "A mathematical framework for the definition of topological relationships," in *Fourth International Symposium on Spatial Data Handling*, (K. Brassel and H. Kishimoto, eds.), (Zurich, Switzerland), pp. 803–813, July 1990.

[8] M. Egenhofer and J. Sharma, "Topological relations between regions in R^2 and Z^2," in *Third International Symposium on Large Spatial Databases, SSD '93*, (Singapore), pp. 316-336, Lecture Notes in Computer Science, vol. 692, New York, NY: Springer-Verlag, June 1993.

[9] R. Franzosa and M. Egenhofer, "Topological spatial relations based on components and dimensions of set intersections," in *International Society for Optical Engineering, SPIE's OE/Technology '92-Vision Geometry*, (R. Melter and A. Wu, eds.), (Boston, MA), vol. 1832, pp. 236–246, Nov. 1992.

[10] O. Guenther and A. Buchmann, "Research issues in spatial databases," *SIGMOD RECORD*, vol. 19, no. 4, pp. 61–68, 1990.

[11] N.W. Hazelton, L. Bennett, J. Masel, "Topological structures for 4-dimensional geographic information systems," *Computers, Environment, and Urban Systems*, vol. 16, no. 3, pp. 227–237, 1992.

[12] J. Herring, "The mathematical modeling of spatial and non-spatial information in geographic information systems," in *Cognitive and Linguistic Aspects of Geographic Space*, (D. Mark and A. Frank, eds.), pp. 313–350, Dordrecht: Kluwer Academic Publishers, 1991.

[13] S. de Hoop and P. van Oosterom, "Storage and manipulation of topology in Postgres," in *Third European Conference on Geographical Information Systems, EGIS '92*, (Munich, Germany), pp. 1324–1336, Apr. 1992.

[14] D. Mark, "Cognitive image schemata and geographic information: relation to user views and GIS interfaces," in *GIS/LIS '89*, (Orlando, FL), pp. 551–560, Nov. 1989.

[15] S. Pigot, "Topological models for 3d spatial information systems," in *Autocarto 10*, (D. Mark and D. White, eds.), (Baltimore, MD), pp. 368–392, March 1991.

[16] D. Pullar and M. Egenhofer, "Towards formal definitions of topological relations among spatial objects," in *Third International Symposium on Spatial Data Handling*, (D. Marble, ed.), (Sydney, Australia), pp. 225–242, Aug. 1988.

[17] N. Roussopoulos, C. Faloutsos, and T. Sellis, "An efficient pictorial database system for PSQL," *IEEE Transactions on Software Engineering*, vol. 14, no. 5, pp. 630–638, May 1988.

[18] P. Svensson and H. Zhexue, "Geo-SAL: a query language for spatial data analysis," in *Advances in Spatial Databases—Second Symposium, SSD '91*, (O. Günther and H.-J. Schek, eds.), pp. 119–140, Lecture Notes in Computer Science, vol. 525, New York, NY: Springer-Verlag, Aug. 1991.

[19] L. Talmy, "How language structures space," in *Spatial Orientation: Theory, Research, and Application*, (H. Pick and L. Acredolo, eds.), pp. 225–282, New York, NY: Plenum Press, 1983.

ESPRIT Project EP6881
AMUSING

Paolo Giulio Franciosa* Maurizio Talamo*

1 General description

The ESPRIT Project EP6881 *AMUSING* (Algorithms, Models, User and Service INterfaces for Geography) is a three year research project started on November 1, 1992. The project involves the following sites: University of Rome "La Sapienza", (Italy); (main partner) University of L'Aquila, (Italy); Algotech s.r.l., Rome, (Italy); INRIA/CNAM, Paris, (France); IGN, Paris, (France); Fernuniversität Hagen, (Germany); National Technical University Athens, (Greece); Technical University Wien, (Austria); ETH Zürich, (Switzerland).

The general objective of AMUSING is to define the main features and principles of a next generation architecture for Geographical Information Systems (GIS). Such an architecture must offer a variety of powerful functionalities in terms of efficiency of data management (both at the physical and logical levels), effectiveness of interaction with users and external services, composability of such services (i.e. data repositories, specific GIS tools, etc.).

Hence, the project does not refer to one specific research theme (such as HCI, data models, computational geometry), but aims at integrating such areas, which are usually separated in research programs, with respect to a specific application field such as GIS.

We envision the following topics as corresponding to mandatory pieces of such an architecture:

- Identifying and coping with typical user requirements for different classes of users (end users, application developers, etc.) of such an architecture.
- Providing advanced data models and query languages for spatial information. Emphasis is given in the project both to the definition of original data models and languages and to the application of existing approaches (extended relational, object-oriented, algebraic, etc.) to geographical data modeling and querying. Interaction with spatial data management systems by means of a powerful set of user-system communication mechanisms are also investigated.
- Providing tools for the cooperation between various application software systems and different spatial data management services. This should range from tight integration to loose federation.
- Presence of efficient mechanisms for the management of data with multiple representations and for the maintenance of intersystem dependencies.
- Providing efficient data structures and algorithms for the management, i.e. querying and manipulating, of large amounts of geometric information. Particular emphasis is given to robust general purpose data structures supporting a wide range of query types.

*Università di Roma "La Sapienza", Dipartimento di Informatica e Sistemistica, via Salaria 113, I–00198 Roma, Italy.

Results obtained within the project are assessed by means of prototyping and case studies. This activity includes building new prototypes, exchanging existing prototypes among AMUSING partners, extending and evaluating them, applying them to case studies conducted in cooperation with end-users for real-life applications.

The final goal is to define a common platform for the development of GIS's providing:

- precise formalisms and tools to analyse and measure requirements of particular applications;
- software modules for the generation of user interfaces and softwa re tools for the mapping of user level specifications to the system data model;
- a data model that offers elegant modeling and treatment of networ ks and heterogeneous collections of objects, spatial data types based on a clean underlying framework of cell topology, and facilities to keep trac k of the lineage of geometric information;
- a system architecture implementing this data model with networks a nd heterogeneous collections in query processing, containing spatial data types (and allowing addition of new types), structures for the representation of partitions, and well defined interfaces to external computation services;
- a library of data type implementations and spatial data structure s that are offered as "extension packages" and can be linked directly in to the extensible system architecture.

It will be attempted to view prototype systems to be built within the scope of the project as subsystems of the above platform, and to adapt them to this environment. As a whole, such a platform will offer a basis for the implementation of GIS that far exceeds the current state of the art.

The final goal of the research activity in AMUSING is defining the structure of a "target" architecture of a GIS. Its basic subsystems are:

- interaction environment: will allow the definition and use of a user interface;
- spatial object manager: has to manage spatial objects and their multiple representation at a physical level;
- transformation manager: allows the creation of new objects by applying algorithms for computing functions on sets of spatial objects.
- a data model manager: considering the AMUSING integrative approach, it should allow a uniform view on heterogeneous data both comi ng from traditional databases or defined in the framework of our system.

The following three sections describe the activities in which the project is articulated: user requirements, data models and prototyping, and more theoretical aspects on data structures and algorithms—showing the partners participating in each activity, results achieved so far, and research directions to be followed.

2 User requirements and case study

Partners involved: Univ. of Rome, Algotech, Univ. of Wien and IGN.

We investigate the special properties of spatial data; these investigations concentrate on parameters related to: (i) data volume, (ii) distribution, size, and topological characteristics, (iii) modelling of geometry, and (iv) changes during use and maintenance of the data set.

The objectives are: to define parameters that characterize spatial data with associated methods for measuring them, apply these methods to a variety of real data sets to determine particular values and typical ranges for these measurements, and to apply solutions provided by the project to real-life case studies (to be conducted in cooperation with end-users).

These investigations, resulting in a characterization of geographic data, will have a strong influence on most other fields within the project, such as data modeling, system architecture, and data structures. They also provide criteria for the selection of a suitable data structure for a particular application and for tuning solutions defined within the project to specific applications.

User requirements and their impact on complex application design have been captured thanks to the interaction with two end users, such as Tiber Bacin Authority (Rome), which is mainly interested in integrating time varying hydrological settings and static scenarios describing orography and administrative boundaries [3] and IGN (National Geographic Institute, Paris), which deals with sets of spatial objects whose geometric attributes change over time.

3 System architecture: data models, languages and prototyping

Partners involved: Univ. of Rome, Univ. of L'Aquila, Algotech, INRIA/CNAM, IGN, Univ. of Hagen, Univ. of Athens, Univ. of Wien, and ETH Zürich.

The overall objectives of this activity are: to define extensions of the relational model to deal with geographical information, to model network structures, to define methods for representing heterogeneous collection of objects, to analyse modeling problems regarding interaction between mathematical models and spatial DBMS, to investigate topological data models.

Further studies on adequacy and evaluation of existing advanced data models and languages for the representation and management of geographical data have been performed [7, 8, 13]. The aim of these studies is to identify the most promising approaches for geographical data modeling, the definition of high level manipulation primitives of the DML, and support for efficient implementation of the modelling primitives.

In this framework we study how to model structural properties of complex structures like graphs and their spatial embeddings, planar decompositions, etc.

Several systems have been designed and developed within the framework of AMUSING: CARTECH [2] is a geographical DBMS, developed by Algotech, based on a model for the management of complex data. The focus of the model is managing abstract data types and constraints at the same time, allowing the manipulation and query of spatial objects. GRAM [1] is a graph data model and query language, which deals with graph databases.

A unified environment is currently being developed that allows the evaluation of DBMSs' performances while dealing with heterogeneous data types represented by means of different physical data structures. Our main goals are: the creation of an embedded framework inside a high level declarative language; the building of powerful tools for the evaluation and comparison of the performances of spatial data structures; the definition of a library of complex spatial constraint, which can be referenced in geographic data handling applications and mathematical models.

4 Data structures and algorithms

Partners involved: Univ. of Rome, Univ. of Hagen, Univ. of Athens and ETH Zürich.

The main objective is to extend classical spatial data structures and/or to design innovative data structures in order to manage complex spatial object. A complex spatial object is either a spatial object whose shape has some structure or an object that cannot be represented in a constant amount of memory.

Research activities follow four directions:

- Strengthen results on the representation of sets of simple spatial objects, e.g. intervals in one dimension or rectangles in the plane, in order to better classify formal properties of classical data structures. For this objective we achieved an optimal result for the representation of set of intervals on secondary storage

[11]. Benefits deriving from the use of high dimensional point structures for representing spatial objects are presented in [14], while an algorithm for optimal splits in R-trees is provided in [5]. A clear overview on problems related to rectangles, such as separation and partition of sets of isothetic rectangles, is given in [6].

- Evaluate the behaviour of classical data structures for spatial objects. Since worst case and "uniform distribution based" probabilistic analysis seem to be not satisfying, we concentrate on the identification of new "instance sensitive" criteria, (e.g. competitive analysis) in order to compare the performances of data structures with respect to adequate adversaries. A deep analysis of performances of range queries on classical spatial data structures in shown in [15], and in [12] performances of intersection queries are studied on instances typical of geographical data sets.

- Characterize relations between geometric objects, evaluating their impact on the design of spatial data structures. Currently some results have been achieved by looking at the general problem of representing an order relation, which models a large number of both topologic and metric properties on sets of geometric objects. This problem has been approached as a dominance problem on directed graphs. Static and dynamic solutions to reachability problems have been proposed in [17, 18], whose space-time tradeoff is related to the size of the transitive reduction of the graph, thus improving current bounds for the general case.

- Manage spatial objects represented at different levels of detail,

  in order to support complex operations on very complex shapes. More precisely our goal is to avoid managing exact (thus heavy) representations of objects any time an approximated representation is sufficient for computing the required functions (e.g. as in multiple scale representations, see [16]). Current results are concerned with convex hull computation [9] on static points and with the problems related to moving objects [10]. Approximation of polygonal objects by means of simpler shapes is treated in [4].

References

[1] B. Amann and M. Scholl. GRAM: a graph datamodel and query language. In *Proc. of ECHT'92*, Milano (Italy), December 1992.

[2] E. Apolloni, F. Arcieri, S. Ercoli, E. Nardelli and M. Talamo. Un Modello di riferimento per l'interazione con sistemi per la gestione di dati geografici (A reference model for interaction with geographic DBMS's). In *Proc. of Convegno Nazionale su Sistemi Evoluti per Basi di Dati*, 14–15 June 1993, Gizzeria Lido, Italy.

[3] E. Apolloni, G. Batini and M. Talamo. Parallel alarm models: assessment of flooding impact in the Tevere river basin. In *Proc. of AM/FM European Conference "Spatial Management in a Europe Without Borders"*, October 13–15, 1993, Strasbourg (France).

[4] B. Becker, P.G. Franciosa, S. Gschwind, S. Leonardi, T. Ohler, and P. Widmayer. Approximating a set of objects by two minimum area rectangles. Università di Roma "La Sapienza," Technical Report 21.92, November 1992.

[5] B. Becker, P.G. Franciosa, S. Gschwind, T. Ohler, G. Thiemt, and P. Widmayer. Enclosing many boxes by an optimal pair of boxes. In *Proc. 9th Sympos. Theoret. Aspects Comput. Sci., Lecture Notes in Computer Science* 577, pages 475–486, Springer-Verlag, 1992.

[6] F. d'Amore. *Algorithms for partitioning and management of sets of hyperrectangles*. PhD thesis, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza," May 1993.

[7] B. David, L. Raynal, G. Schorter and V. Mansart. GeO2: Why objects in a geographical DBMS? In *Proc. 3rd Int. Symposium in Large Spatial Databases (SSD)*, Singapore, June 1993 in: David Abel, Beng Chin Ooi (eds.): Advances in Spatial Databases, *Lecture Notes in Computer Science* 692, Springer Verlag.

[8] B. David, L. Raynal, G. Schorter and V. Mansart. GeO2: Object-Oriented Contribution for a geographical DBMS. In *Proc. of DEXA'93*, Prague, September 1993.

[9] P.G. Franciosa, C. Gaibisso, G. Gambosi and M. Talamo. A convex hull algorithm for points with approximately known positions. Submitted to Internat. J. Comput. Geom. & Appl.

[10] P.G. Franciosa, C. Gaibisso and M. Talamo. Optimal algorithms for the maxima set problem for data in motion. Technical Report RAP.03.92, Università di Roma "La Sapienza," Dipartimento di Informatica e Sistemistica, February 1992. Also presented at Workshop on Computational Geometry, March 12–13, 1992, Utrecht, The Netherlands.

[11] R. Giaccio and M. Talamo. A general framework to deal with sets of intervals. In *Proc. of GULP'93*, June 16–18, 1993, Gizzeria Lido, Italy.

[12] J. Nievergelt, P. Widmayer. Guard files: stabbing and intersection queries on fat spatial object. *The Computer Journal*, 1993.

[13] D. Papadias. Spatial relation-based representation systems. In Proc. of *European Conference on Spatial Information Theory (COSIT'93)*, Elba, Italy, September 1993.

[14] B.U. Pagel, H.W. Six and H. Toben. The transformation technique for spatial objects revisited. In *Proc. of 3rd Int. Symposium in Large Spatial Databases (SSD)*, Singapore, June 1993 in: David Abel, Beng Chin Ooi (eds.): Advances in Spatial Databases, Lecture Notes in Computer Science No. 692, Springer Verlag, 73–88.

[15] B.U. Pagel, H.W. Six, H. Toben and P. Widmayer. Towards an analysis of range query performance in spatial data structures. In *Proc. Twelfth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, Washington, D.C., May 25-28, 1993, 214–221.

[16] P. Rigaux, M. Scholl and A. Voisard. A Map Editing Kernel Implementation: Application to Multiple Scale Display. In *Proc. of European Conference on Spatial Information Theory (COSIT'93)*, Elba, Italy, September 1993.

[17] M. Talamo and P. Vocca. *Overcoming the $O(n^2)$ space*time bottleneck for directed graph reachability problem*. Submitted to SIAM J. on Computing.

[18] P. Vocca. *Space-time trade-offs in directed graph reachability problem*. PhD thesis, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza," May 1993.

1994 International Conference on Applications of Databases ADB-94

June 21-23, 1994
Vadstena Monastery Hotel, Sweden

CALL FOR PAPERS

SPONSORED BY:
The Swedish Board for Technical Development (NUTEK), Hewlett-Packard, Softlab AB, Ellemtel AB, GMD. In cooperation (requested) with ACM, AFCET, AICA, GI, IEEE, IFIP, INRIA, VLDB Endowment Inc.

SCOPE

Successful database research and development has made many general database systems available. The industrial offerings range from traditional large mainframe systems, through workstations & PC servers, to massproduced $100 personal systems. Users, application designers, and the software industry strive to get the most from this technology. Database researchers, and developers try hard to further satisfy application needs. The conference aims at developing synergy between these communities. It is intended as the forum to explore innovative applications of databases, and innovative database services for specific applications.

PAPER SUBMISSION

We solicit research, industrial and user's papers, panel proposals, and tutorial proposals. Papers should be original, not exceeding 6.000 words, (25 double spaced pages), and with abstract. Accepted papers will be published by Springer Verlag. All submissions will be notified of acceptance or rejection by April 1, 1994. Camera ready copy will be due for the conference proceedings by April 14, 1994.

Six copies of the submission, including e-address or fax number, should be sent by February 14, 1994 to:
ADB-94 Secretariat, email: adb94@ida.liu.se, Anne Eskilsson, Dept of Computer and Information Science, Linköping University, S-581 83 LINKÖPING, Sweden.

LOCATION

The town of Vadstena is located about 300 km. south of Stockholm, bordering lake Vättern, the 2nd largest lake in Sweden. Vadstena is a medieval town with cobbled streets, an ancient abbey and castle. It is near the town of Linköping, center for high-tech industry, and home of one of the largest universities in Sweden. The conference site is the monastery originally built as a royal palace in 1258, pre-dating Stockholm. It is the oldest secular building in Sweden. The conference week includes the longest day of the year, when there is light all night in Vadstena and midnight sun in northern Sweden. June 24 is also the national 'Midsummer' holiday in Sweden, when many ethnic celebrations take place. June/July is about the best time for vacations in Scandinavia.

General Topics of interest will include but are not limited to:

APPLICATIONS
General Enterprise Management
Financial and Business Decision Support, Reporting, Spreadsheets
Interpersonal Communication: Datebooks, Contacts, Meetings, Email
Electronic Documents, Publishing, and Libraries
Application Development: CUI, GUI, CASE, Development Libraries, Interfaces, and Utilities
Telecommunications: Switching & Billing, On-Line Services, National Information Infrastructure
Personal Assistants for Mobile Users
Manufacturing: CAD, CAM, Signal Processing, Robotics
Consumer Services: Insurances, Banking, Real-Estate, Travel, Sales, Marketing & Distribution
Mathematics: Formal Calculus, Numerical Analysis, Statistics, Computational Geometry
Applied Sciences & Engineering: Geography, Chemistry, Aeronautics, Biotechnologies
Healthcare
Law & HumanitiesTraining & Education
Leisure & Entertainment

SERVICES
Interoperability of databases, and with legacy applications
Visual Database Languages
Databases on multicomputers, parallel computers, and data highways
Mobile databases
Agents and Mediators
New storage media and technologies
Text, Image, Sound, and Video in databases
Object Management and Libraries
Data migration to databases
Database performance, benchmarking, and tuning
Large Scale Information Retrieval and Mining
Time, Event Management, and Monitoring
Customization of database manipulations
Use of standards: SQL, ODBC, OLE, DDE, WOSA, DRDA, CORBA, DCE....
DBMSs on new OSs, FMSs, and Transaction Managers: Windows NT, OS2, Chorus, Mach, AFS, Encina, Tuxedo,.....
Security

ORGANIZATION:
General Chair
Tore Risch (U Linköping, Sweden)
Organizing Committee
Chair: Anders Törne (U Linköping, Sweden)
Ingrid Nyman (U Linköping, Sweden)
Anne Eskilsson (U Linköping, Sweden)
Program Chair
Witold Litwin (U Paris 9, v. HPL & U Berkeley)
Tutorial Program:
Umesh Dayal, (HPL, USA)
Panel Program
Ming-Chien Shan, (HPL, USA)

Program Committee
American V-Chair: Dan Fishman (HPL, USA)
European V-Chair: Erich Neuhold (GMD-IPSI, Ger.)
Far East V-Chair: Ron Sacks-Davis (RMIT, Aus.)

America
Rakesh Agrawal (IBM)
Jürgen Annevelink (HPL)
John Carlis (U Minnesota)
Amelia Carlson (Sybase)
Dimitrios Georgakoupoulos (GTE)
Tomasz Imielinski (Rutgers U)
Joseph L. Koszarek (Boeing)
Mohamad Ketabchi (Santa Clara U)
Ravi Krishnamurthy (HPL)
Darrel Long (UC Santa Cruz)
Stuart Madnick (MIT)
Rao Mikkilineni (US West)
Marie-Anne Neimat (HPL)
Tamer Özsu (U Alberta)
Lawrence A. Rowe (UC Berkeley)
Steve Rozen (Whitehead/MIT Genome Ctr)
Marek Rusinkiewicz (Houston U)
Hanan Samet (U Maryland)
Arie Segev (UC Berkeley & LBL)
Amit Sheth (Bellcore)
Douglas Terry (Xerox Parc)
Frank Tompa (Waterloo U)
Andrew Whinston (U Texas, Austin)

Europe
Francois Bancilhon (O2 Techn., France)
Horst Biller (Siemens Nixdorf, Germany)
Anders Björnerstedt (Ellemtel Telecom. SL, Sweden)
Janis Bubenko (U Stockholm, Sweden)
Stefano Ceri (P Milano, Italy)
Stavros Christodoulakis (TU Crete, Greece)
Georges Gardarin (U Versailles, France)
Michael Hanani (Bromine Compounds Ltd., Israel)
Genevieve Jomier (U Paris 9)
Leonid Kalinichenko (Russian A. Sc., Russia)
Wolfgang Klas (GMD-IPSI, Germany)
eva Kuehn (TU Vienna, Austria)
Peter Lockemann (U Karlsruhe, Germany)
Robert Roffe (ADB.SA/Intellitic, France)
Felix Saltor (UP Catalunya, Spain)
Marc Scholl (U Ulm, Germany)
Stefan Schneider (GOPAS, Germany)
Michael Schrefl (U Linz, Austria)
Witold Staniszkis (Zeto Rodan, Poland)
Per Svensson (N Def. R. Estbl., Sweden)
Henry Tirri (U Helsinki, Finland)
Patrick Valduriez (INRIA, France)
Jerker Wilander (SoftLab AB, Sweden)
Roberto Zicari (U Frankfurt, Germany)

Far East
Mark Bilger (IBM, Hong Kong)
Anande Deshpande (Persistent Syst, India)
Yahiko Kambayashi (Kyoto U, Japan)
Masaru Kitsuregawa (U Tokyo, Japan)
Ramamohanarao Kotagiri (U Melbourne, Aus.)
Fred Lochovsky (U Sc. and Techn., Hong Kong)
Yoshifumi Masunaga (U Libr. and Inf. Sc., Japan)
Desai Narasimhalu (ISS, Singapore)
Beng Chin Ooi (NU Singapore, Singapore)
John Smith (CSIRO, Australia)
Kunitoshi Tsuruoka (NEC C&C Syst. RL, Japan)
Kyu-young Whang (KAIST, Korea)
Michael Yap (Nat'l Comp. Bd., Singapore)

IMPORTANT DATES
Submissions: February 14, 1994,
Notif. of acceptance: April 1, 1994
Conference: June 21-23, 1994

CALL FOR PARTICIPATION

DATA ENGINEERING

Tenth International Conference on

Data Engineering

February 14-18, 1994

Doubletree Hotel, Houston, Texas

Sponsored by the IEEE Computer Society

IEEE

SCOPE

Data Engineering deals with the modeling and structuring of data in the development and use of information systems, as well as with relevant aspects of computer systems and architecture. The Tenth Data Engineering Conference will provide a forum for the sharing of original research results and engineering experiences among researchers and practitioners interested in automated data and knowledge management. The purpose of the conference is to examine problems facing the developers of future information systems, the applicability of existing research solutions and the directions for new research.

TECHNICAL PROGRAM HIGHLIGHTS

Research Papers Sessions on:

- Disk Storage Management • Management of Distributed Data
- Query Processing • Analytical Modeling • Temporal Databases
- Multidatabase Systems • Knowledge and Rule Management
- Indexing Techniques • Data Mining • Parallel Databases
- Heterogeneous Information Systems.

Invited Presentations:

- Al Aho (Bellcore): "Engineering Universal Access to Distributed Interactive Multimedia Data".
- Gio W ieder hold (ARPA): "From Data Engineering to Information Engineering".

Panel Discussions:

- Mobile Computing
- Business Applications of Data Mining
- Future Database Technologies.

Technology and Application Track:

- Practice-oriented presentations of applications of database technologies.

TUTORIAL PROGRAM HIGHLIGHTS

1. **Active Database System:** Klaus Dittrich (Zurich Univ.) & Jennifer Widom (IBM Almaden), Sunday, Feb. 13 1994, (full day).
2. **New Developments in SQL Standards:** Krishn Kulkarni (Tandem Computers Inc.) and Andrew Eisenberg (DEC), Monday morning, February 14, 1994, (half day).
3. **User Interfaces and Databases:** Prasun Dewan (University of North Carolina), Tuesday Morning, Feb. 15, 1994, (half day).
4. **Multimedia Database Systems:** Arif Ghafoor (Purdue University), Monday Afternoon, February 14, 1994, (half day).
5. **Object-oriented Modeling of Hypermedia Documents:** Wolfgang Klas, Karl Aberer (GMD - IPSI), Monday Morning, February 14, 1994, (half day).
6. **Medical Databases:** Lynn L. Peterson and J. C. G Ramirez (University of Texas), Tuesday Afternoon, Feb. 15, 1994, (half day).
7. **Object-Oriented Systems Development: From Analysis to Implementation:** Gerti Kappel (University of Vienna) and Gregor Engels (Leiden University), Monday Afternoon, February 14, 1994, (half day).

HOTEL RESERVATION INFORMA TION

Call the Houston Doubletree Hotel Post Oak at (713) 961-9300 or toll-free 1-800-528-0444 to make your hotel reservation. To obtain the special conference rate of U.S. $85.00 per night for Single or Double, you must mention you are attending the ICDE-94 conference. The cut-off date for guaranteed guest room reservations is Feb. 1, 1994. If you have any questions on registration, tutorials, or program, please send e-mail to icde94@cs.uh.edu or fax to (713) 743-3335.

REGISTRATION FORM AND FEES SCHEDULE

• ADVANCE (Received by January 10, 1994) •

- **Registration**: Member($290), Non-Member($360), Student($70)
- **Tutorials (Full /Half day):** Member($200/$120), Non-Member($250/$150)

• LATE/ON-SITE (Received after January 10, 1994) •

- **Registration:** Member($320), Non-Member($460), Student($110)
- **Tutorials (Full /Half day):** Member($240/$150), Non-Member($300/$185)

The conference registration fee covers the proceedings, conference reception, and refreshments, but not the banquet. Banquet tickets for the Texas Evening, Thursday Feb. 17, 1994 are $38/ea. Additional reception tickets may be purchased for $30/ea. Payment should be in US dollars ONLY (check drawn on a US bank, International Money Order, or credit card) Please complete this form, and return by Jan. 10, 1994 with your payment (payable to ICDE-94) to:

Dr. Albert Cheng, ICDE-94,
Computer Science Department, University of Houston
Houston, TX 77204-3475, USA, Fax: (713) 743-3335

For information contact the Program Chair, Marek Rusinkiewicz: icde94@cs.uh.edu, (713) 743-3350

Name: _____

Address: _____

_____

_____

E-mail: _____

Fax: _____

•Advance Registration: $_____

•Advance Tutorials: $_____

Please check the tutorial Number(s):

1__ 2__ 3__ 4__5__ 6__ 7__

•Add'l Reception Tickets ($30/ea): $_____

•Banquet Tickets ($38/ea): $_____

•Total Amount Enclosed: $_____ Signature:_____

• Credit Card: o Visa o Mastercard o American Express

• Credit Card Number: _____ • Exp. Date:_____

IEEE Computer Society
1730 Massachusetts Ave, NW
Washington, D.C. 20036-1903

Non-profit Org.
U.S. Postage
PAID
Silver Spring, MD
Permit 1398