Query Languages for Geographic Information Systems

Three papers by:

Max J. Egenhofer and Andrew U. Frank

National Center for Geographic Information & Analysis
and
Department of Surveying Engineering
University of Maine
Orono, ME 04469

National Center for Geographic Information & Analysis/NCGIA

Report 90-12

Preface and Acknowledgements

The three papers contained in this report show three different perspectives of interactive query languages for geographic information and analysis, an emphasis of the NCGIA's Research Initiative No. 2 on "Languages of Spatial Relations." They are published as a technical report to make them accessible to readers as a whole so that they can make comparisons between the different approaches.

The first paper, "Concepts of Spatial Objects in GIS User Interfaces and Query Languages," was presented at GIS/LIS '89 in Orlando. It investigates the requirements for a GIS query language from a user's perspective. The second paper, "Spatial SQL: A Query and Representation Language," shows how such a GIS query language can be designed as an extension of the well-known relational query language SQL. Finally, the third paper, entitled "LOBSTER: Combining AI and Database Techniques for GIS," is a reprint of an article published in Photogrammetric Engineering and Remote Sensing, Vol. 56, No. 6, June 1990, pp. 919-926. It reports on the implementation of and experience with a prototype of a spatial query language based upon the Prolog programming language.

We believe that the papers complement each other and, as such, give an fairly broad overview. The framework for this work has been set in a complementary technical report in which the initiative leaders explain, among other things, the issues and relevance of spatial query languages.

Concepts of Spatial Objects in GIS User Interfaces and Query Languages*

Max J. Egenhofer
National Center for Geographic Information and Analysis
and
Department of Surveying Engineering
University of Maine
Orono, ME 04469, USA
MAXOMECAN1.bitnet

Abstract

An interactive query languages is an essential part of a GIS user interface to allow the user to pose ad-hoc queries. Traditional database query languages, such as SQL or Query-byExample, have provide insufficient services for the treatment of spatial properties. Deficiencies observed include the requirement to model spatial data in terms of relations, the users' need of a sophisticated mathematical knowledge in order to express queries with spatial conditions, and the lack of representations of query results in forms other than text.

This paper presents a methodology for the design of spatial query languages which are embedded into a human interface. The methodology is based upon the definition of *objects* and *operations* available at the conceptual level of the user interface. This object-oriented approach is complemented by the selection of appropriate techniques to interact with spatial objects rendered on a screen.

1 Introduction

Current database query languages, which are well-suited to treat alphanumeric data, provide insufficient services as spatial query languages. Due to the specific properties of spatial data, query languages for spatial information systems are more complex than languages dealing only with text. For instance, a spatial query language must include tools for all the essential operations to inquire about spatial and non-spatial data.

For an overall satisfying solution it is first necessary to consider the role of a query language within a spatial information system from a user's perspective. A query language must support those concepts that humans use in their daily life when they deal with space. As such, the user interface in which the query language is embedded becomes crucial, linking closely the design and development of spatial query languages to user interface considerations.

In this paper, those concepts will be investigated which users apply when they request and analyze spatial information. First, an overview is given over user concepts, interface styles, and interaction techniques as they have been developed in Computer Science and Psychology. Human interface considerations for spatial information systems will be reviewed. In section 4 a collection of user concepts of spatial data handling will be compiled. Many of them deal with the possible graphical representation of spatial data. The paper ends with a summary of its significant results which are presented in the form of a comprehensive list of eight requirements for a spatial query language.

2 User Interfaces

The principal purposes of an information system are twofold: (1) providing the service of assisting users and (2) supporting the users in their decisions. The effectiveness of these tasks depends upon the way users may request information and the form in which this information is presented to them. Hence, information systems are under a specific obligation to meet user requirements.

The *user interface* of an information system enables humans to request and receive information. Its design influences how easily users may interact with a system and how quickly they understand the results presented. The interface of an information system as the integrating part of all particular applications should hide internal details-how data are stored, composed, or decomposed, etc.-so that users are enabled to concentrate on their work or pleasure [Shneiderman 1987].

A query language is an essential part of a user interface. It enables the users to communicate with a computer system by articulating their instructions in a form that can be "understood" by a system. For this goal, formal languages are necessary.

* This research was partially funded by grants from NSF under No. IST 86-09123 and Digital Equipment Corporation. The support from NSF for the NCGIA under grant number SES 88-10917 is gratefully acknowledged.

Frequently, the query language is considered only a tool to select data of interest. Most conventional query languages provide often complex methods for retrieval of data and formulations of logical constraints upon data. Structured query languages with question-answer dialog dominate the interaction with conventional, purely alphanumeric information systems, such as SQL [Chamberlin 1976], QUEL [Stonebraker 1976], and Query-By-Example [Zloof 1977].

Most of these query languages are based upon a comprehensive mathematical model, such as relational algebra or calculus, and user interface aspects are restricted to considerations about the representation of and interaction with data within the mathematical framework. As a consequence, users of such languages manipulate tables and their content. While this approach is appropriate for applications in which users conceive data in tabular form, it is unfit for applications in which users do not naturally use the mental model of tables. In general, spatial data are associated with geometry, map representation, etc., but not with tables.

2.1 Conceptual User Level

Investigations under interdisciplinary efforts in Computer Science and Psychology have been concentrating on improving the design of interfaces with computers. Their goal is to accommodate the interaction between users and system to human thinking, rather than enforcing the users to organize their thoughts as dictated by a system. Humans are the focus of this process which intents to meet the concepts humans use as closely as possible. "Human interface" is the keyword for the studies of techniques and methods which have significantly improved the design of interfaces.

One theoretical foundation of user interfaces is the *Seeheim model* [Green 1985] which differentiates three conceptual parts of a user interface: (1) the presentation, (2) the dialog, and (3) the application interface. If dialog management is intermixed with other purposes, independent changes of either the application or the dialog are difficult to achieve, prone to error, and often even impossible.

Obviously, input and output play a key role in presentation and dialog. A sophisticated model of interactive input and output from the perspective of a programming language was proposed by Shaw [Shaw 1986]. This model abandons the archaic view of feeding data into a program and getting a result out at the end. Instead, a software system is considered a permanently running machine to which "probes" can be attached to examine the state of the machine. This approach provides a framework for design principles which are otherwise difficult to achieve, such as the display which should reflect the current program state at any time; or the separation of the input-output interface from the main program.

New methodologies for the design of human-computer interfaces have been developed which influence today's interface design and try to respond to the needs of the humans who use a computer system. Their goal is to make software systems faster to learn and easier to use.

2.2 Interface Styles

Under these cross-disciplinary efforts, several principles and methods fundamental for the current stream of designing human interfaces have been developed. The use of metaphors, for example, is central to the process of inventing graphical user interfaces. Metaphors may be employed for presenting, explaining, and making familiar new capabilities or functionality by exploiting the humans' association of an object with a familiar domain. The trash can on the Macintosh interface, for instance, is a part of the well-known desktop metaphor. It symbolizes the place where to discard, the analogy of which is applied in the software environment to deleting files.

Another famous design principle is WYSIWYG. The "what- you-see-is-what-you-get" concept manifests that the interface reflects at all times exactly the state of the system, such that the user's view of the system coincides with its actual state. Popular WYSIWYG systems are text editors and spreadsheets which update the user view with every user command entered.

In order to facilitate the design of effective user interfaces, many tools have been introduced to help generate interfaces in the design phase. These tools go beyond the paper and pen design [Myres 1986]; however, most of them deal with purely lexical representation, e.g., as lists or tables, or focus on the integration of popular interaction techniques, such as menus, panels [Freburger 1987], and direct-manipulation devices [Shneiderman 19831. Generally, graphics are used in a traditional manner, providing drawings to the users that they can view [Tanner 1986] [Gutfreund 1987].

3 User Interfaces for Spatial Information Systems

Within the last decade, the design of spatial information systems has been explored in a bottom-up manner and only little attention has been paid to the interface through which a user views such a system. It is a common, yet unfortunate opinion that the interface is "something to be done after the design and the implementation has been completed." Such "user interfaces" are cosmetic

enhancements which barely help to make these systems "user friendly." The reverse process-first designing the user interface, followed by the implementation of the system-is only rarely pursued.

A remarkable exception is the query language MAPQUERY [Frank 1982] the design of which considered the interaction between a user and spatial data. A few other interface papers report on the use for methods and techniques to geographic applications [Chong 1988] or demonstrate how tools like Hypercard may be used for fast implementations [Gould 1988]. Only little consideration is paid to the role of the user interface as a query language or the conceptual differences arising from the treatment of spatial and non-spatial data [Egenhofer 1988].

A somewhat broader view of user interfaces for spatial information systems is provided from some database researchers who investigated the incorporation of display and interaction with spatial data. The design and implementation of user interfaces which are primarily graphics oriented have been acknowledged as challenging tasks [Chu 1983]. User interface considerations for the acqui sition of spatial data have demonstrated that the use of advanced concepts and techniques may overcome inherent problems in simulating the traditional ways humans work with spatial data [White 19881.

Some considered the user interface a crucial component of an information system which has the capability to represent query results graphically. Herot at al. reported on a system in which a user traverses through a window, a two-dimensional surface on which objects are displayed in iconic form [Herot 1980]. The system provides the user navigational aids by showing a "world-view map" on which the current position may be identified. By zooming in and out, the level of detail displayed may be controlled. This action is supported by a "hierarchy map" on which users may learn possible routes to take from one surface to another.

4 User Concepts in Spatial Query Languages

All the principles and techniques of user-machine interfaces contribute to a much broader goal, the user's *conceptual model*, which establishes the relationship between the abstract concepts presented by the software and the user's tasks. An example for a sound conceptual model is the design concept of a *desktop*. On a desktop, documents are organized in folders and drawers, and several sheets of paper may be distributed across the desk. Sheets may be moved around, pulled in front of or pushed behind others. The implementation of this familiar concept on workstation screens is supported by icons symbolizing drawers and folders, windows simulating paper sheets, and the mouse-in combination with WYSIWYG and direct manipulation-for selecting or dragging documents across the screen. Users familiarize quickly with this style of interface, because it appeals to them almost naturally.

Considerations about user interfaces for spatial information systems must start with the development of the user's conceptual model of spatial data. Fundamental differences between managing non-spatial and spatial data were recognized by several authors. Herot et al. describe them from the perspective of the database management system [Herot 1980]. They state that conventional database management systems organize information in a number of abstract name spaces which are referenced by providing values to be matched, while spatial data management systems organize information in concrete two-dimensional geometric spaces. Dangermond argues that relational techniques are not suitable for map data due to the topological relationships which make geometric modifications more complex, involving more than a single record in a update [Dangermond 1988]. The development of a user's model of spatial data must start with the definition of objects and operations available at the conceptual level and the selection of appropriate techniques to interact with the objects on the screen.

4.1 Spatial Objects

The manipulation of spatial data through query languages has been hindered by insufficiently powerful abstractions of spatial data at the user interface level. Conventional data can be modeled in terms of integers, reals, and strings as humans perceive them as such. For instance, a banking account may consist of a user number (an integer), the user's name (a string), and the current balance (an integer). The corresponding operations upon such a banking account can be reduced to operations upon simple data types, e.g., modifying the balance by withdrawing a certain amount may be implemented as the subtraction of one integer number from another. Likewise, the constraints for operations are concise, e.g., the operation *withdraw* cannot be executed unless resulting balance remains above a certain amount.

Communicating with spatial data at this same level of abstraction provides significant difficulties. Users would be required to deal with too low a level of abstraction of spatial data. Instead, two fundamental concepts of communication with spatial data are introduced:

- the view of spatial data in the form of *complex objects*; and

- the interaction with spatial objects through pertinent *operations*.

The view of objects and operations has a sophisticated foundation in the object-oriented model. Within the last few years, the object-oriented approach has been promoted as a suitable model for complex situations [Dittrich 1986], such as CAD/CAM [Buchmann 1985] [Sidle 1980], office automation [Harder 1985], and GIS/LIS [Frank 1984]. It supports an advanced user concept of data handling, i.e., the treatment of arbitrarily complex objects [Lorie 19831. The definition of object types is not restricted to a specific framework, such as the form of a relational table in the relational data model, nor is it limited to a few, pre-defined data types. Object types may be complex aggregates, such as molecules in chemistry [Batory 1984], solids in CAD/CAM [Lee 1983], circuits in Very Large Scale Integration (VLSI) systems [McLeod 1983], or geographic objects [Egenhofer 1989a].

Complex objects play a central role in a user's concept of spatial data. A land parcel, for example, is a geographic object that consists of some property description, such as its land use and its owner, and a geometry description. The latter may be related to or consist of several other objects, such as the boundaries separating the parcel from its neighbors or the monuments defining where the boundary runs. From a user's perspective the decomposition is insignificant and as such should be hidden.

The object-oriented paradigm offers another concept, the combination of an object type with its pertinent operations in a comprehensive unit, which may serve as a framework for a high-level treatment of spatial objects. In order to access or manipulate objects, a set of operations is provided which allow users to perform their desired tasks without any specific knowledge about the internals of the particular objects. For example, the operation *getNeighbors* may be defined for objects of type "parcel," the use of which does not require from a user any particular knowledge about the details of the structure or implementation of parcels.

Such operations are of outstanding importance for the actions users are executing upon spatial objects from a query language. The operations to be carried out upon spatial data are not all as simple as the ones in the previous banking example. Frequently, complex restrictions and constraints apply, for instance, that the coverage of all parcels forms a complete partition.

4.2 Graphical Display

The ability to present spatial data graphically is an obvious difference between spatial and conventional information systems. The *display* of query results in graphical form is the most natural form to analyze spatial data. Graphics are for humans much faster to understand than any lexical representation with the same amount of data. Compare only the simple representations of a listing of coordinate tuples and a plot of the corresponding positions (figure 1).

x-coordinate	y-coordinate
2119.75	21135,51
4309.24	18590.28
1389.92	21983.92

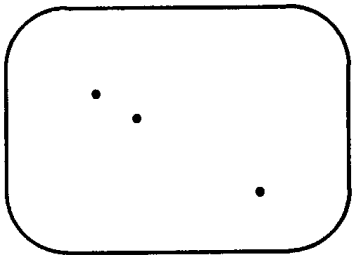


Figure 1: Alphanumeric vs. graphical representation of points.

By exploiting a second dimension, many spatial relationships-otherwise difficult to perceive get revealed, such as information about neighborhood, inclusion, or intersection. Even humans with strong intellectual abilities show extreme difficulties to interpret these spatial relationships from numbers only; however, most humans can easily decide by looking at the graphical representation whether two lines intersect or which is the closest point to a given line. In figure 1 the graphical representation reveals almost immediately for any human that one point lies between the other two, while the pure coordinate data does not. Besides rendering the geometry of spatial objects, the graphical display allows for the visualization of issues which are related to spatial objects and represented in the database as non-spatial attributes. Hence, graphical representation of query results is a key characteristic of a spatial query language.

4.3 Selection By Pointing

The formulation of a query is based on either the user's knowledge, or information which was provided from the system. A map on a graphical display, for example, provides a large variety of information which the user can exploit for upcoming queries. Conceptually, this information can be separated into the objects displayed and the information about their spatial distribution across the two-dimensional surface.

The goal of the interaction with a spatial information system is to use the representation available to the user as reference in upcoming queries. Conventional dialog is completely typed from a keyboard. An extended dialog using pointing devices *for selection by pointing* promotes the usage of query results as reference in upcoming queries and reduces the use of the keyboard [Friedell 1982]. Selection by pointing is a major object-oriented feature of graphical presentation because users can select, what they see, exploiting spatial information which is only visible on drawings.

A second type of information conveyed through a graphical representation is the spatial distribution of the objects across the 2-dimensional surface. Upon this surface, a user may perform operations, such as wandering around (*pan*) or viewing a specific part in a more detailed representation (*zoom*). The latter is a common technique in geographic analysis, when a situation is often first viewed from a greater distance before the region or object of interest is investigated more closely. For such iterative approaches, a graphical scene serves as reference to the selection of the following rendering. A typical query with reference to a sub-area is, "Show all streets and towns within this region" and the user selects the region from a rendering on the screen. This process can be supported from the dialog using direct-manipulation methods to select the area of concern.

4.4 Directing the Graphical Representation

Interactive-graphic presentation is powerful for mapping systems because the content of maps can be quickly modified. While traditional maps are a static product-once the map is printed it cannot be updated unless overwriting it with some pencil-mapping with spatial information systems promotes immediate manipulations on drawings. Objects can be added to, removed from, or modified on an existing map without the need to redo the drawing from scratch. This concept of modifying the content on the screen is different from updating the content of the database. Display updates provide another view over the same data set, while database updates actually modify the objects stored. In a WYSIWYG environment such database updates imply screen updates if the modified objects lie within the area displayed; however, reversely screen updates by adding or removing query results do not modify the database.

Display modifications require that the user has tools to manipulate the composition of a map. Unlike conventional systems which treat each result as an entirely new representation and do not refer to earlier results, several interactive-graphic drawings are often overlaid with each other [Frank 1987] or one "thematic layer" is removed from another. These are powerful processes in spatial information systems, combining spatial data of arbitrary sources to a composition that appears optimal for a user's application.

The possible combination of one query result with the results of one or more previous queries gives rise to a dynamic and user-friendly interaction. Directing the graphical display to manipulate drawings is essential to overcome the static character of traditional representation or paper maps. It must be supported by the five graphical presentation types in an interactive-graphic information system:

- *New* refreshes the viewport before drawing the next picture.
- *Overlay* adds the result of the current query to the existing picture.
- *Remove* erases the result of the current query from the existing picture.
- *Intersect* selects all those objects that are both on the display and in the query result.

The semantics of these operations can be derived from conventional set operations. If the current display and the query result are considered as sets then the following analogies yield:

- Overlays correspond to the set union combining the set of currently displayed objects with the query result.
- Intersect corresponds to the set intersection of the set of displayed objects with the query result.
- Remove is the set difference operation cancelling all objects of the query result from the set of objects currently displayed.
- New yields the intersection of the query result with the empty set.

Further display operations [Scholl 1989] which cannot be expressed in terms of pure set theory are the geometric selections of *windowing* and *clipping* and the *superimposition* and *highlighting* [Egenhofer 1989b] of query results.

4.5 Appropriate Graphical Context

Graphical representations frequently require the display of *context*, i.e., information which was not explicitly asked for, but which is necessary to interpret a query result in its spatial location.

The interpretation of graphical representation is extremely sensitive to the context and environment in which it is shown. Unlike lexical representation, it is often not sufficient to draw only those objects that were asked for. Imagine a query "Show the town of Orono," where the result is only a point in the center of the screen [Frank 1982] as shown in figure 2.

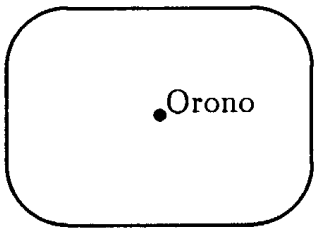


Figure 2: An insufficient answer to the query "Show the town of Orono."

Sophisticated graphical representation must consider the selection of appropriate context which depends upon the purpose of the drawing, the scale, and the data density. The users must be provided with tools to select the most efficient context for their maps from the user interface.

A reasonable answer to the previous query would have required a context in which the position of the point could have been spatially interpreted. For example, by showing the borders of the state of Maine, users familiar with the geography of Maine are given sufficient information to locate Orono (figure 3).

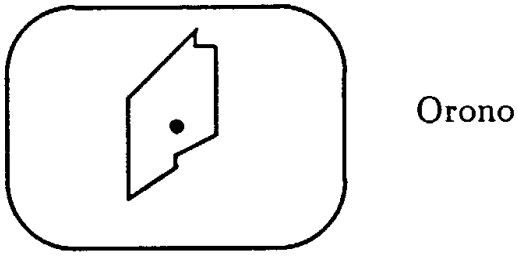


Figure 3: The query "Show the town of Orono" with minimal context.

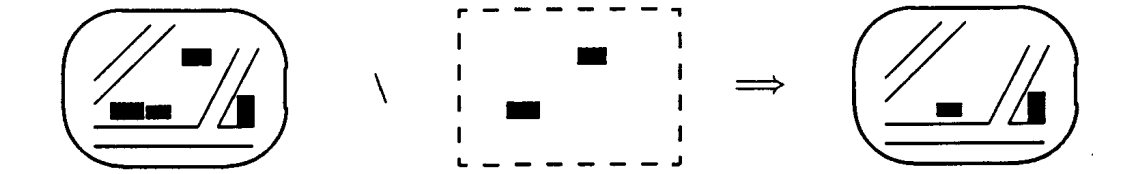
Spatial context may be considered as an intelligent application of query combinations. While the user asks for a particular object, the system completes the result by "overlayine" another query result. This combination facilitates the interpretation of the user query.

4.6 Content Examination

All previous concepts were based upon operations to be executed upon the data set in the form of queries and representations, yet none dealt with the results. *Examination* complements these concepts. It assists users in the interaction with the results presented.

An operation complementary to the display operations is the examination of the *content* of the display. The combination of multiple query results in a single rendering allows for dynamic changes on the drawing. Composing the results of several queries makes it more difficult to keep track of what is actually presented. An essential feature in a query environment with multiple

representation modes is a control mechanism which allows the user to check, after a sequence of queries, what a single query for the current drawing would have been. For example, if a user asks for a map showing all roads and buildings (figure 4a) and subsequently requests to remove from this rendering all residences (figure 4b), thenbesides the roads-only those buildings are represented which are not residences (figure 4c).



(a) All roads and buildings. (b) Residences. (c) All roads and buildings except for residences.

Figure 4: Content examination in a multi-query environment.

4.7 Varying Graphical Representations

Graphical representation is a powerful and effective means to convey spatial information. The techniques used to distinguish different objects from each other are of particular importance if a multitude of spatial data is represented in a single drawing. A utility map, for instance, rendering various utility lines, parcel boundaries, and buildings will be of little value if all objects are represented with black lines of the same style and width. Users are given too few clues about the different meanings of the lines and so they will have difficulties in reading the map.

Another application of various graphical representation techniques is the rendering of the same data in various fashions according to different views and purposes. The view of a road, for example, is different for a surveyor, a transportation company, or a cartographer. One sees it as a parcel, the other as a classified transportation line, another as symbolic line, and for each purpose the road may be represented differently.

The structured data collection of a spatial information system provides a favorable background for flexible renderings if the appropriate techniques are available to monitor the intended conveyance of information. *Varying graphical representation* is the key to this concept. It allows the user to choose the appropriate representation for spatial data in order to present the intended information most effectively and have the user understand it best. Graphical representation of spatial objects can be described by techniques used in traditional cartography, such as colors, patterns, intensity, and symbols [Bertin 1983].

Varying graphical representation may be used as a fundamental object-oriented principle to convey the concept of "classes" in graphics. Classification is the grouping of several objects with similar properties to a common class [Dahl 1966]. The graphical properties are an effective too] supporting the following two concepts:

- By using the same graphical representation for several objects at distinct locations, the user is conveyed the information that these objects are similar.

- By using significantly different representations the dissimilarity of objects can be emphasized.

The following example may illustrate these concepts. Imagine a map with three buildings, two of them represented as filled rectangles, the other as a cross-hatched box. From this representation a user will assume that (1) the two "filled" buildings are similar, i.e., have similar properties, and (2) the "cross-hatched" building has one or several properties which distinguish it from the "filled" buildings. The two buildings may be residences while the other is an office building, or the two may be worth more than $100,000, while the other is less.

4.8 Legend

The interpretation of graphical representations becomes more difficult with a greater variety of object classes to be represented by different symbols, colors, patterns, etc. While the different classes may be easily distinguished, their meaning is sometimes difficult to interpret without a suitable explanation. For example, a map with two different symbols for buildings, one representing residences, another one office buildings, requires an explanation such that the user may draw the intended conclusions (figure 5).

While the distinct representation informs the user about the *existence* of differences between objects the boxes in figure 5 represent two types of buildings-it does not inform the user about their *semantics*, i.e., what the two distinct representations stand for. Unless these semantics are accessible to the user in an understandable form, the graphical representation conveys incomplete information.

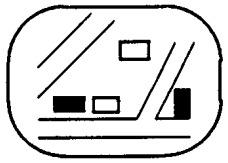


Figure 5: A representation of two types of buildings which requires an explanation of the symbols used to convey the complete information to the users.

The *legend* is the key for the interpretation of drawings. It complements varying graphical representation as the *examination* method. The concept of a legend is adopted from traditional cartography which uses it for the explanation of the symbols used on a map. Essentially, the legend is the dictionary of an interpreter translating from a graphical into an alphanumeric language. For example, the map in figure 5 needs a legend describing that black boxes are residence and white boxes office buildings. Figure 6 shows this translation from graphical into alphanumeric language.

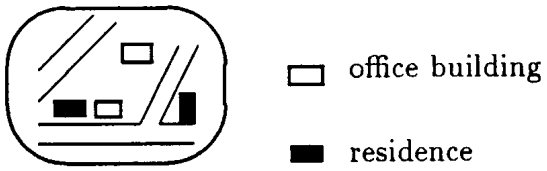


Figure 6: Example of a legend to interpret a drawing.

5 Conclusion

The design of spatial query languages should start with investigations about the concepts which users employ when they deal with spatial data; therefore, this chapter investigated what users perceive when they communicate with spatial information systems and how modern techniques may support the interaction with and representation of spatial data. From studies in human-computer interaction fundamental principles, such as "seeing and pointing," the WYSIV%TYG paradigm, and metaphors were identified as particularly useful concepts for the interaction with spatial data.

A set of eight requirements for a spatial query language was elaborated, each of which represents important concepts in the interaction with spatial data and distinguishes spatial data handling from conventional treatment of non-spatial data. These concepts are:

1. An abstract data type "spatial" with corresponding operations and relationships is needed to provide sufficiently high abstraction of spatial data at the user interface.

2. The display of query results in graphical form as the most natural form to represent spatial data is crucial since it allows for the representation of geometry as well as the visualization of properties which are non-spatial, yet related to spatial objects.

3. An extended dialog using pointing devices for selection by pointing and the direct selection of a sub-area promotes the usage of query results as reference in upcoming queries and reduces the use of the keyboard.

4. The possible combination of one query result with the displayed results of previous queries gives rise to a user-friendly interaction in which renderings are dynamically changed.

5. Graphical representations frequently require the display of context, i.e., information which was not explicitly asked for, but which is necessary to interpret a query result in its spatial location.

6. The examination of the content of a drawing provides important information which is otherwise not accessible for a user in a multi-statement query environment.

7. Varying graphical representation of spatial objects and their parts can be described in terms of colors, patterns, intensity, and symbols.

8. Finally, varying graphical representation allows the production of graphics about which the user may want information. The concept of a legend was introduced as the interpreter between the graphical language on the screen and an alphanumeric language.

References

[Batory 1984] D.S. Batory and A.P. Buchmann. Molecular Objects, Abstract Data Types, and Data Models: A Framework. In: 10th International Conference on Very Large Data Bases, pages 172-184, Singapore, August 1984.

[Bertin 1983] J. Bertin. Semiology of Graphics. The University of Wisconsin Press, Madison, WI, 1983.

[Buchmann 1985] A.P. Buchmann and C.P. de Celis. An Architecture and Data Model for CAD Databases. In: 11th International Conference on Very Large Data Bases, pages 105-114, Stockholm, Sweden, 1985.

[Chamberlin 1976] D. Chamberlin, M. Astrahan, K. Eswaran, P. Griffiths, R. Lorie, J. Mehl, P. Reisner, and B. Wade. SEQUEL 2: A Unified Approach to Data Definition, Manipulation, and Control. IBM Journal of Research and Development, 20(6):560-575, November 1976.

[Chong 1988] D. Chong and G. Dudly. Considerations in the Development of a GIS User Interface. In: GIS/LIS '88, pages 665-672, San Antonio, TX, November 1988.

[Chu 1983] K.-C. Chu, J. Fishburn, P. Honeyman, and Y. Lien. VDD-A VLSI Design Database System. In: Engineering Design Applications, Annual Meeting, Database Week, pages 25-37, San Jose, CA, May 1983.

[Dahl 1966] O.-J. Dahl and K. Nygaard. SIMULA-An Algol-based Simulation Language. Communications of the ACM, 9(9):671-678, September 1966.

[Dangermond 1988] J. Dangermond. A Technical Architecture for GIS. In: GIS/LIS '88, pages 561-570, San Antonio, TX, November 1988.

[Dittrich 1986] K. Dittrich. Object-Oriented Database Systems: The Notation and The Issues. In: K. Dittrich and U. Dayal, editors, International Workshop in ObjectOriented Database Systems, Pacific Grove, CA, pages 2-4, IEEE Computer Society Press, Washington, D.C., 1986.

[Egenhofer 1988] M. Egenhofer and A. Frank. Towards a Spatial Query Language: User Interface Considerations. In: D. DeWitt and F. Bancilhon, editors, 14th International Conference on Very Large Data Bases, pages 124-133, Los Angeles, CA, August 1988.

[Egenhofer 1989a] M. Egenhofer and A. Frank. Object-Oriented Modeling in GIS: Inheritance and Propagation. In: AUTO-CARTO 9, Ninth International Symposium on Computer-Assisted Cartography, pages 588-598, Baltimore, MD, April 1989.

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

[Frank 1982] A. Frank. MAPQUERY-Database Query Language for Retrieval of Geometric Data and its Graphical Representation. ACM Computer Graphics, 16(3):199207, July 1982.

[Frank 1984] A. Frank. Requirements for Database Systems Suitable to Manage Large Spatial Databases. In: International Symposium on Spatial Data Handling, Zurich, Switzerland, August 1984.

[Frank 1987] A. Frank. Overlay Processing in Spatial Information Systems. In: N.R. Chrisman, editor, AUTO-CARTO 8, Eighth International Symposium on ComputerAssisted Cartography, pages 16-31, Baltimore, MD, March 1987.

[Freburger 1987] K. Freburger. RAPID: Prototyping Control Panel Interfaces. In: OOPSLA '87, pages 416-422, Orlando, FL, 1987.

[Friedell 1982] M. Friedell, J. Barnett, and D. Kramlich. Context-Sensitive, Graphic Presentation of Information. ACM Computer Graphics, 16(3):181-188, July 1982.

[Gould 1988] M. Could and S. Love. An Easily-Customized User Interface for GIS and Map Display. In: ACSM-ASPRS Fall Convention, Virginia Beach, VA, 1988.

[Green 1985] M. Green. Report on Dialogue Specification Tools. In: G.E. Pfaff, editor, Computer Graphics Forum, Springer Verlag, New York, NY, 1985.

[Gutfreund 1987] S.H. Gutfreund. ManiplIcons in ThinkerToy. In: OOPSLA '87, pages 307-317, Orlando, FL, October 1987.

[Harder 1985] T. Harder and A. Reuter. Architecture of Database Systems for NonStandard Applications (in German). In: A. Blaser and P. Pistor, editors, Database Systems in Office, Engineering, and Science, pages 253-286, Springer Verlag, New York, NY, March 1985.

[Herot 1980] C. Herot, R. Carling, M. Friedell, and D. Kramlich. A Prototype Spatial Data Management System. ACM Computer Graphics, 14(3):63-70, July 1980.

[Lee 1983] Y.C. Lee and K.S. Fu. A CSG Based DBMS for CAD/CAM and its Supporting Query Language. In: Engineering Design Applications, Annual Meeting, Database Week, pages 123-130, San Jose, CA, May 1983.

[Lorie 1983] R. Lorie and W. Plouffe. Complex Objects and Their Use in Design Transactions. In: D. DeWitt and G. Gardarin, editors, ACM Engineering Design Applications, pages 115-121, May 1983.

[McLeod 1983] D. McLeod, K. Narayanaswamy, and K.V. Bapa Rao. An Approach to Information Management for CAD/VLSI Applications. In: Engineering Design Applications, Annual Meeting, Database Week, pages 39-50, San Jose, CA, May 1983.

[Myres 1986] B.A. Myres and W. Buxton. Creating H ighly-I nter active and Graphical User Interfaces by Demonstration. ACM Computer Graphics, 20(4):249-258, August 1986.

[Scholl 1989] M. Scholl and A. Voisard. Thematic Map Modeling. In: A. Buchmann, 0. Giinther, T. Smith, and Y. Wang, editors, Symposium on the Design and Implementation of Large Spatial Databases, Santa Barbara, CA, pages 167-190, Springer-Veriag, New York, NY, July 1989.

[Shaw 1986] M. Shaw. An Input-Output Model for Interactive Systems. In: Human Factors in Computing Systems, CHI '86, pages 261-273, Boston, MA, April 1986.

[Shnelderman 1983] B. Shneiderman. Direct Manipulation: A Step Beyond Programming Languages. Computer, 16(8):57-69, August 1983.

[Shneiderman 1987] B. Shneiderman. Designing the User Interface: Strategies for Effective Human-Computer Interaction. Addison-Wesley, Reading, MA, 1987.

[Sidle 1980] T.W. Sidle. Weakness of Commercial Database Management Systems in Engineering Applications. In: 17th Design Automation Conference, 1980.

[Stonebraker 1976] M. Stonebraker, E. Wong, and P. Kreps. The Design and Implementation of INGRES. ACM Transactions on Database Systems, 1(3):189-222, September 1976.

[Tanner 1986] P. Tanner, S. MacKay, D. Steward, and M. Wein. A Multitasking Switchboard Approach to User Interface Management. ACM Computer Graphics, 20(4):241-248, August 1986.

[White 1988] R.M. White. Applying Direct Manipulation to Geometric Construction Systems. In: N. Magnenat and D. Thalmann, editors, New Trends in Computer Graphics: Computer Graphics International '88, pages 446-455, Geneva, Switzerland, May 1988.

[Zloof 1977] M.M. Zloof. Q uery- by- Example: A Database Language. IBM Systems Journal, 16(4):324-343, 1977.

Spatial SQL: A Query and Representation Language*

Max J. Egenhofer
National Center for Geographic Information and Analysis
and
Department of Surveying Engineering
University of Maine
Orono, ME 04469, USA
MAXOMECAN1.bitnet

Abstract

Recently, attention has been focused on spatial databases, which combine conventional and spatially related data, such as Geographic Information Systems, CAD/CAM, or VLSI. A language has been developed to query such spatial databases. It recognizes the significantly different requirements of spatial data handling and overcomes the inherent problems of the application of conventional database query languages. The spatial query language has been designed as a minimal extension to the interrogative part of SQL and distinguishes from previously designed SQL extensions by (1) the preservation of SQL concepts, (2) the high-level treatment of spatial objects, and (3) the incorporation of spatial operations and relationships. The novel approach is the syntactical separation of database query and display specification into the query and the representation language, respectively. Users can ask standard SQL queries to retrieve non-spatial data based on non-spatial constraints, use Spatial SQL commands to inquire about situations involving spatial data, and give instructions in GRL to manipulate or examine the graphical representation. Separately formulated queries and display specifications are integrated during query processing to provide an optimized query solution in a single pass. This paper explains the semantics and syntax of Spatial SQL, the spatial query language, and the Graphical Representation Language GRL.

I Introduction

Recently, attention has been focused on spatial databases which combine conventional and spatially related data. The need for spatial query languages has been identified in several different application domains such as Geographic Information Systems (GIS) [Frank 1984b], image databases [Chang 1981], remote sensing [Lohman 1983], CAD/CAM systems [Lee 1983], or VLSI design [McLeod 1983]. The usage of standard query languages for spatial data handling has been hindered by the lack of appropriate language support for spatial data. Query languages such as SQL [Chamberlin 1976], Quel [Stonebraker 1976], or Query-by-Example [Zloof 1977] are insufficient for spatial data because they address only the particular properties of lexical data.

Some attempts have been made to apply existing (relational) query languages to spatial data as well. These languages treat spatial data as integers and strings in relational form, so that users need an understanding of implementation details of spatial data, which is comparable to the seeing the treatment of reals as bit strings. Examples are GEO-QUEL [Berman 1977] [Go 1975], a QUEL extension, Query-by-PictorialExample [Chang 1980], a Query- by- Example extension, or a Quel extension for image processing [Embley 1986].

Spatial data have additional properties, such as *geometry* and *graphical representation*, which the user must be able to address in a query language. The importance of spatial relationships and operators for spatial query languages has been recognized [Frank 1982) [McKeown 1983] [Roussopoulos 1985] and the relational algebra has been extended with spatial operations to a Geo-Relational Algebra [Gdting 1988]. But a spatial query language must be more than the simple equation*: Relational Query Language + Spatial Relationships = Spatial Query Language*. More functionality was added to languages, such as SQUEL, which combines QUEL and graphical representation with which interactive communication is encouraged [Herot 1980], or GEOBASE [Barrera 1981] which has a flavor of inheritance and propagation of attributes and values in generalization and aggregation of spatial hierarchies, respectively. Similar extensions which are based upon SQL [Herring 1988] [Ingram 1987] [Roussopoulos 1985] [Sikeler 1985] will be reviewed at more detail in section 2.3.

In an earlier paper, we analyzed the requirements for a spatial query language describing user concepts about spatial data and their integration into a human interface [Egenhofer 1988b]. In this paper we demonstrate how these concepts can be translated into the syntax of a spatial query language. In the context of spatial data handling, a query language is seen as broader than only a solution for the *retrieval* of data. Here, the appropriate *representation* of spatial data is also considered to be part of a query language.

* This research was partially funded by grants from NSF under No. IST 86-09123, Digital Equipment Corporation under Sponsored Research Agreement No. 414, and Intergraph Corporation. Additional support from NSF through the NCGIA under grant SES 88-10917 is gratefully acknowledged.

Instead of developing an entire new query language from scratch, an existing database query language is extended with spatial concepts. As the host language, SQL was chosen, and the resulting spatial query language is called *Spatial* SQL. Similar attempts of extending an existing query language with new concepts have been undertaken in other areas, for instance with the design of the temporal or historical query languages TQUEL [Snodgrass 1987], TOSQL [Ariav 1986], and HSQL [Sarda 1990] as respective extensions of QUEL and SQL, and SQL dialects for date and time [Date 1988] and dynamic complex objects [Lorie 1988]. The particular combination of the description of retrieval and representation in a single query language distinguishes the design of Spatial SQL from other languages. The retrieval part is based upon SQL, the standard query language for relational databases [ANSI 1986]. It is completed by the *Graphical Representation Language* (GRL), a comprehensive tool for the description of the graphical display. The goal of Spatial SQL is to provide higher abstractions of spatial data in a query language by incorporating concepts closer to the humans' thinking about space and the present paper explains the semantics and syntax. Performance considerations will not be addressed here. Also, the architecture of this system [Egenhofer 1989b], the embedding of Spatial SQL into a human interface [Egenhofer 1988b] and query processing and optimization [Hudson 1988], have been described elsewhere.

The remainder of this paper is organized as follows: Section 2 evaluates previous SQLbased query languages for spatial databases against a list of requirements for a spatial query language and identifies particular deficiencies in the treatment of the graphical representation of query results. The architecture of an SQL-based spatial query and representation language is discussed in section 3. Sections 4 and 5 present the syntax and semantics of the retrieval language Spatial SQL and the Graphical Representation Language GRL, respectively. An example in section 6 shows the use of Spatial SQL and GRL. Finally, in section 7 our conclusions are presented.

2 Spatial Query Languages

Spatial SQL is based upon the relational database query language SQL [Chamberlin 1976] which was originally designed as the query language for System R [Astrahan 1976], IBM's relational database management system. The framework of an SQL query is the SELECTFROM-WHERE clause which corresponds to the three operations of relational algebra *projection*, *Cartesian product*, and *selection*, respectively. The other two fundamental relational operations, *set intersection* and *set union*, may be performed explicitly between two SQL queries. Aggregate functions, such as *sum*, *minimum*, or *average*, extend the power of relational algebra [von Baltzingsloewen 1987] by calculating a single value from a set of tuples.

2.1 Guidelines for an SQL Extension

The decision to exploit SQL as the backbone for a spatial query language was driven by the recognition of efforts to standardize SQL as *the* database query language [ANSI 1986]. The reason for extending an existing query language, as opposed to developing a new one, was also influenced by the recognition that spatial databases contain both spatial and nonspatial data which will be the subject of user queries. Three fundamental categories of queries in a spatial information system can be distinguished [Barrera 19811:

- Queries exclusively about spatial properties, e.g., "Show all towns that are split by a river."
- Queries about non-spatial properties, e.g., "How many people live in Orono?"
- Queries which combine spatial and non-spatial properties, e.g., "List all neighbors of the parcel located at 30 Grove Street."

It is crucial for spatial query languages to provide syntactical means for all three categories. Traditional query languages already provide a solution for the formulation of non-spatial queries. For example, the query for the number of people living in Orono may be expressed as the following standard SQL query:

```
SELECT      population
FROM        town
WHERE       name = "Orono";
```

A spatial extension to a non-spatial query language must preserve all its alphanumeric functionalities to allow the user to pose non-spatial queries appropriately.

The premise of the design of this SQL extension has been to retain the concepts of the host language. Likewise, the characteristic structure of the language with the SELECT-FROM-WHERE clause should stay untouched.

The following concepts of standard SQL were specifically regarded:

• Every query result is a relation.

• The SELECT-FROM-WHERE construct is the framework of every query.

• Predicates in the WHERE clause are formulated upon attributes.

Every extension of SQL as a superset of standard SQL has to live with the flaws of SQL. For this particular work, Codd's recent critique of SQL [Codd 1988] is not relevant; however, other limitations in the design of SQL make the language difficult to extend for certain spatial concepts. For instance, the tabular representation of the result, implied for every interactive SQL query, is inappropriate for spatial applications which frequently require graphical results. The spatial query language presented in this paper will not improve SQL as such and should not be considered a "better SQL." Instead, it will be shown how easy-or how difficult-it is to integrate certain spatial concepts into SQL.

The second guideline deals with the number of spatial additions to be made to the syntax of SQL. The current structure of SQL with the SELECT-FROM-WHERE block (and possibly additional GROUP-BY-HAVING clauses) is already considered to be complex enough to use [Luk 19861. Additional clauses for the treatment of new concepts are undesirable. Although such extensions-adding a clause for graphical representation, another one for graphical context, etc.-might appear as a possible solution, they would certainly further increase the users' problems of formulating syntactically correct queries.

2.2 Requirements for Spatial Query Languages

Investigations of conventional query languages identified eight crucial requirements for a spatial query language which are not covered by conventional systems [Egenhofer 1988b]. They are:

1. An abstract data type spatial with corresponding operations and relationships is necessary so that users may treat spatial data at a level independent from internal coding, such as the representation of spatial objects on the level of x-y coordinates [Roussopoulos 1988].

2. The display of query results in graphical form, as the most natural form to analyze spatial data, allows for the representation of the geometry of spatial objects and the visualization of issues which are related to spatial objects but represented in the database as non-spatial attributes.

3. An extended dialog using pointing devices for selection by pointing and direct selection of a subscene promotes the usage of query results as a reference in upcoming queries and reduces the use of the keyboard [Friedell 1982].

4. The possible combination of one query result with the results of one or more previous queries gives rise to a dynamic interaction.

5. Graphical representations frequently require the display of context, i.e., information which was not explicitly asked for but which is necessary to interpret a query result in its spatial location.

6. Induced by the combination of multiple query results in a single rendering, users need control mechanisms to check the content of a drawing.

7. Varying graphical representation of spatial objects and their parts described in terms of colors, patterns, intensity, and symbols [Bertin 1983] is by far more complex than the description of the format of a table and requires dedicated language tools.

8. A descriptive legend is needed which reflects a summary of the object classes displayed together with their particular representation.

2.3 SQL-Based Query Languages for Spatial Databases

Several proposals have been made to make SQL applicable as a query language for spatial databases, the most significant extensions of which will be reviewed subsequently.

A proposal with minimal extensions to SQL is GEOQL. This language adds to the standard definition the concept of geometry in terms of the bounding lines of spatial objects; spatial operators between geographic objects; and windows [Ooi 1989a].

Sikeler proposed two extensions [Sikeler 1985]: (1) the treatment of spatial relations, and (2) a picture list which manages the graphical output. The spatial relations are separated from the relations among non-spatial data by means of an additional clause (WITH LOCATION) in which predicates with spatial relationships, such as next-to, can be formulated. The picture list is added to the attribute list of the SELECT statement. In order to distinguish which parts to print and which parts to draw, the qualifier GRAPHIC is introduced. Spatial results of subqueries are tagged with the postfix loc. This proposal assumes a standard graphical format for each relation which is insufficient for most queries. The treatment of relations instead of attributes in the WITH LOCATION clause and the GRAPHIC qualifier does not conform with the general concept of SQL which requires that predicates are defined upon attributes.

PSQL (Pictorial SQL) is an SQL extension tailored to raster image.. processing [Roussopoulos 1985]. Each spatial object is extended by an attribute loc which is a pointer to the picture of the object. This attribute is referenced in the SELECT clause for graphical output and in a specific clause for treating spatial relations. PSQL adds two clauses to the SELECT-FROM-WHERE construct: (1) AT specifying the area to treat, and (2) ON describing a predefined output format (picture list), such as a specific map type. The picture list allows juxtaposition (synthesis) of dissimilar information of the same area from different sources. The introduction of two additional clauses in PSQL makes the formulation of queries unnecessarily complicated. Complex queries which involve the logical combination of spatial and non-spatial predicates may require repeated AT-WHERE clauses. In a recent update of this language the logical consequence was drawn and the AT clause was merged with the WHERE clause [Roussopoulos 1988]. A predefined representation is very useful, but it needs a language to manipulate the representation.

The SQL-based query language for the Geographic Information System KGIS [Keating 1987] is a combination of syntax extensions in the SELECT-FROM-WHERE clauses and a command set outside of SQL [Ingram 1987]. The spatial relationships *distance, overlay, overlap,* and *adjacent* are added to the WHERE clauses, and spatial relations are extended by the attributes *area, perimeter,* and *length.* Objects on the graphic screen can be identified via a PICK routine. Further extensions include (1) the definition of a context map which is added to the query result as a default background, (2) the definition of a window which describes the area of the graphical display, and (3) the specification of the output type (e.g., REMOVE). These features are evaluated as additional clauses outside of the SELECTFROM-WHERE query.

Another proposal for extensions to SQL [Herring 1988] pursues an object-oriented approach, giving up some SQL principles. For instance, the FROM clause is cancelled and the SELECT clause treats entire relations represented through object identifiers. Spatial extensions are Boolean operators for topological concepts, such as *intersect* or *adjacent,* spatial attributes, such as *area* or *centroid;* and "derivation operators" determining parts of spatial objects, such as *boundary,* or compositions, such as *union. To* provide for higher-level and more complex constructs, macro facilities are introduced for predicates. These macros can be used as shortcuts in the WHERE clause. A significant contribution is the attempt to define the spatial relationships, operations, and attributes formally, using mathematical terms from topology and set theory.

The spatial capabilities of these SQL extensions are summarized in Table I where the languages are analyzed against the eight requirements for a spatial query language.

Requirements	GEOQL	Extended SQL	PSQL	KGIS	TIGRIS
Spatial ADT	+	$+^{12}$	+	$+^{2}$	$+^{2}$
Graphical Representation	+	+	+	+	+
Selection By Pointing	+	–	–	+	–
Result Combination	–	–	–	$+^{3}$	–
Context	–	–	$+^{4}$	+	–
Content Examination	–	–	–	–	–
Display Manipulations	–	–	$+^{4}$	–	–
Legend	–	–	–	–	–

Table 1: Evaluation of the four SQL extensions. "+" stands for "fulfill" and "–" for missing.

3 Separating Retrieval and Display Instructions

The production of a map with a single instruction may be of interest [Ehrich 1988], however, the primary goal of the interaction with graphical renderings in a spatial information system is making dynamic changes rather than producing static products. Users are expected to (1) pose several queries in a row the results of which more often change the content of the rendering than its graphical representation style and (2) edit the graphical rendering frequently by modifying only the graphical "parameters" of objects already displayed. In such an environment a single, standard display layout is insufficient for complex graphics, such as maps. On the other hand, the storage of map graphics is inappropriate because the exact rendering depends upon the scale, area of display, and content of a drawing, and must be recomputed for each actual display. For example, screens may become overcrowded [Frank 1981] or labels may be placed outside of the visible screen area [Frank 1984a] (Freeman 1987].

The integration of a full display description into the query language would make each user query unnecessarily complex and long. In lieu of packing the entire request into a single statement, it appears to be beneficial to separate each instruction into several smaller and controllable units. Three types of instructions are distinguished:

- the actual *user query* specifying the retrieval of the set of data to be displayed,
- additional queries, called *display queries*, necessary to separate query results into more detailed sets, each to be displayed in an individual format, and
- the actual *display description* specifying how to render the data.

Here, it is proposed to group these three types of instructions into two languages so that users can separately describe the content of a drawing-what to retrieve-and its appearance how to display the query result. The separation into a *retrieval language* and a *representation language* releases the user from the need of describing the graphical representation with each query. Instead, a display environment is maintained that manages the state of the representation and applies it to the query result. Query and display specifications then interact closely: the representation language describes on the one hand what to do with the result such as, "Show the buildings in the query result (if any) as black squares." On the other hand, the representation language may require expressions similar to a regular query for a detailed description such as, "Show the buildings in the query result which are of type residence as red squares and the commercial buildings as blue squares." The similarity to actual user queries implies that the query language may serve as part of the Graphical Representation Language.

Our particular approach consists of Spatial SQL, a spatial query language designed as an extended SQL for the retrieval of spatial and non-spatial data, and the Graphical Representation Language GRL, a dedicated language for the description of the graphical representation which was designed as a superset of the spatial query language. Fig. 1 shows the three standard situations of a user interacting with GRL and Spatial SQL: a user modifies the display environment with one or several GRL commands which will be effective with the next query asked (Fig. 1a); a user formulates a query in Spatial SQL and the result is presented according the current state of the display environment (Fig. 1b); and a user updates the display environment with a GRL command and requests to update the current display before asking the next query (Fig. 1c). Query and display instructions are combined in a non-procedural way, i.e., users describe the display style and formulate the query, and the system finds the most effective way for integrating and executing a user query and display queries.

The concept of Spatial SQL allows experienced SQL users to continue applying the popular query language for inquiries upon non-spatial data in the familiar way. Only queries that include spatial properties must use the extensions of Spatial SQL. Users familiar with SQL are expected to learn this SQL dialect quickly because it preserves the general concepts and structure of standard SQL and only a few additions are made to the semantics and syntax.

---

[1] Only spatial relationships.
[2] No data definition.
[3] Only for context.
[4] As part of the picture list in the ON-clause.

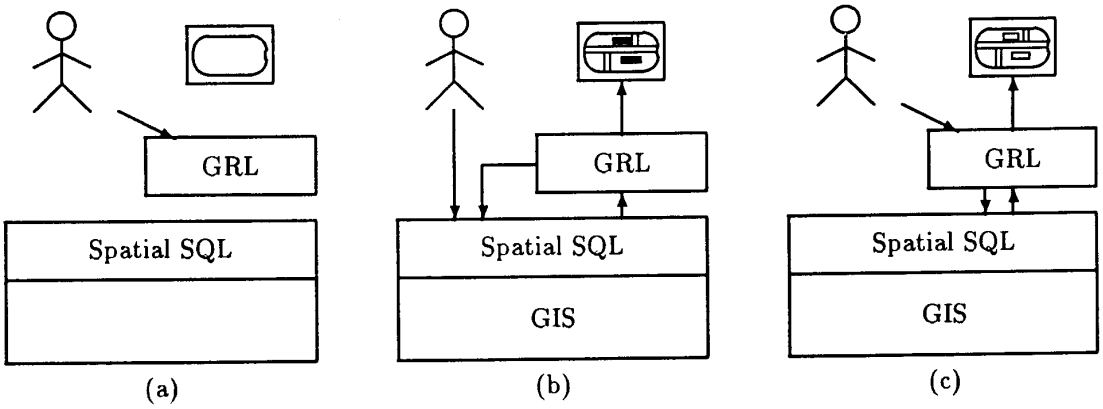


Figure 1: Interaction with Spatial SQL and GRL: a user (a) sets the display environment through GRL, (b) asks queries with Spatial SQL, and (c) updates the graphical representation through GRL.

4 Spatial SQL

In Spatial SQL, the domains of the relational calculus are extended by the new domain *spatial* to provide a high-level abstraction of spatial data at the user interface. Besides the spatial domain and its dimension-dependent specializations, the necessary spatial relationships and a method to refer to objects on drawings by pointing at them with a direct manipulation device are introduced. The semantics and syntax of these extensions are presented in this section.

To date, only subsets of a spatial algebra [Gilting 1988] [Tomlin 1990] and formal definitions of spatial relationships [Egenhofer 1989a] have been developed; therefore, this part of Spatial SQL is extensible so that new spatial operations can be added into this conceptual framework.

4.1 Spatial Domain

It has been demonstrated that the use of "pure" SQL for some spatial queries is cumbersome and syntactically very complex [Westlake 1990]. The treatment of spatial data as abstractions higher than integers or strings is necessary for an appropriate treatment of geometry. While an operation, such as distance between two points, may be familiar to a large community and its implementation might be commonly known, other operations or relationships are by far more complex-or subtle. Mathematically less sophisticated users cannot be expected to acquire all this mathematical knowledge before asking a query with some geometric criterion.

In Spatial SQL, the domains in relational calculus are extended by four spatial domains *spatial_0*, *spatial_l*, *spatial_2*, and *spatial_3*-for 0, 1, 2, and 3-dimensional spatial objects, respectively. The domains are generalized to a dimension-independent domain *spatial*. An attribute over a spatial domain will be referred to as a *spatial attribute*, and a relation with a spatial attribute will be called a *spatial relation*. The properties of spatial relations are significantly different from the properties of the standard relations with integer numbers or character strings. Spatial relationships, for instance, refer to spatial concepts such as topology and metric [Egenhofer 1990c] [Kainz 1990]; however, they are considered as selection criteria, similar to traditional predicates, and can be used similarly to conventional relationships. Operations upon spatial attributes can be grouped into three classes: (1) unary operations accessing a spatial property of a tuple, (2) binary operations calculating a value among two tuples, and (3) binary relationships determining whether a spatial relationship holds among two tuples.

4.2 Unary Spatial Operations

A unary spatial operation can be considered a function upon a spatial attribute. Important spatial functions are those which determine the dimension of an object, its boundary, and its interior. The *dimension* is 0, 1, 2, and 3 for points, lines, areas, and volumes, respectively. *Boundary* determines the bounding faces of an n-dimensional object *0*, i.e., all object parts in the boundary of 0 [Egenhofer 1988a]. Complementary to boundary, *interior* calculates the interior faces, i.e., all those object parts that are not in the boundary of 0. The specializations of these operations determine bounding and interior faces of a specific dimension, e.g., an area has the operations *boundingNodes* and *boundingEdges* for the determination of 0- and 1-dimensional bounding object parts, respectively.

For I- and 2-dimensional objects, the set consists of the operations *boundary, boundingNodes, boundingEdges, interior, znteriorNodes, interiorEdges*, and *interiorAreas*.

A second set of unary spatial operations deals with arithmetic operations. They depend upon the dimension of an object. This set of operations consists of *length*, for a 1-dimensional object, and *area* and *volume* for 2-D and 3-D objects, respectively. More complex attributes can be derived as the combination of topological and arithmetic properties, such as the *perimeter* of a polygon which is defined as the sum of the lengths of the boundingEdges. Other unary operations, such as *extreme coordinates* [Cox 1979], *complement* [Burton 1979], and *convex hull* [Gilting 1988], fit into this concept and may be easily incorporated into Spatial SQL.

4.3 Binary Spatial Operators

Binary spatial operators are defined similarly to the arithmetic operators for atoms of conventional domains, such as addition or multiplication. Two binary spatial operations are introduced in Spatial SQL which map from two spatial attributes to a real number: (1) *distance* and (2) *direction* [5]. Distances can be added, subtracted, multiplied with a scalar, and compared for equality, order, and strict order. Directions can be compared with the same operators and subtracted to yield *angles.* Aggregate functions, such as *minimum* and *average,* can be formulated upon distances and directions.

A prefix formulation for spatial operators, like `distance (city.geometry, highway. geometry)`, is more natural and preferred over the conventional infix form of arithmetic operators. Spatial operators can occur in any SELECT clause in association with two spatial attributes, or in any WHERE clause as part of a non-spatial predicate. Though spatial operators are conceptually different from aggregate functions, they are incorporated into Spatial SQL at the same locations.

4.4 Spatial Relationships

A *spatial relationship* is a relationship between two spatial attributes. Spatial relationships conform with traditional binary relationships and result in a Boolean value. Hence, they can be immediately applied as predicates in the WHERE clause of SQL. Spatial SQL also preserves the infix form of traditional SQL predicates for spatial predicates. Similar to Standard SQL, which overloads relationships like less than or equal, spatial relationships are defined upon the generalized spatial data type *spatial*. The actual implementation is invisible to the user unless a constraint is violated, such as a void predicate, "point contains line."

Binary topological relationships are based upon the set intersections of the boundaries and interiors of the two targets [Egenhofer 1989a] [Egenhofer 1990d]. For example, the neighborhood relationship between two areas can be expressed in terms of the four boundary and interior intersections (Table 2).

$\partial A \cap \partial B$	$\neg\emptyset$
$A^\circ \cap B^\circ$	$\emptyset$
$\partial A \cap B^\circ$	$\emptyset$
$A^\circ \cap \partial B$	$\emptyset$

Table 2: Specifications of the topological relationship *neighbor* between two objects A and B in terms of the intersections of boundaries (∂) and interiors ($^\circ$).

These specifications are dimension independent and, therefore, apply to any two objects of arbitrary dimensions. Fig. 2 shows prototypical examples for the topological relationships most commonly used in geographic applications: *disjoint, meet, overlap, inside/contains, covers/coveredBy,* and *equal*. They are included in Spatial SQL as convenience operations releasing users from the fundamental mathematical knowledge necessary to formulate spatial queries with topological constraints and to provide a higher-level abstraction of geometric concepts. Depending on various orderings, details can be expressed about topological relationships. Possible spatial order relations are *left/right, north/south*, or *over/under.*

[5] Strictly speaking, the types are a positive real number for distances and a range domain (0.. .359°59'59") for directions.

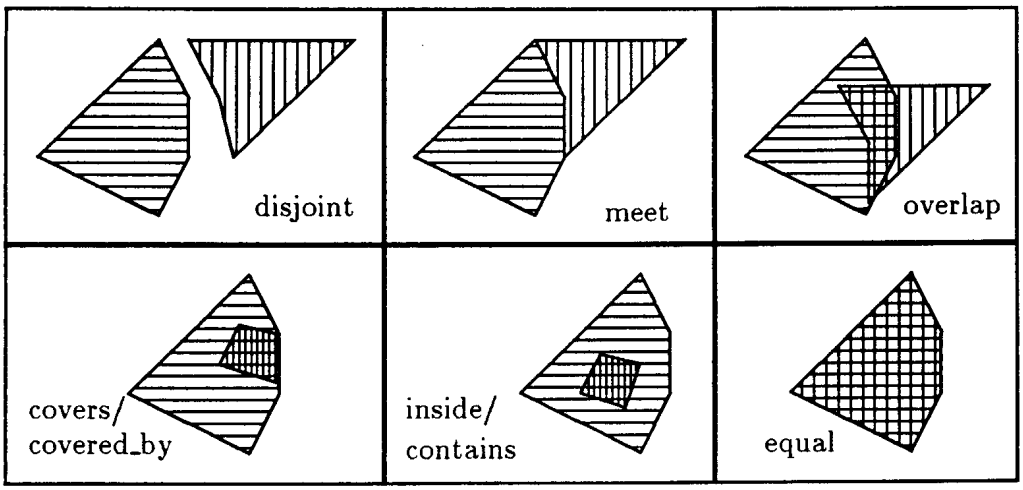


Figure 2: Examples of the relationships (a) disjoint, (b) meet, (c) overlap, (d) covers/covered_by, (e) inside/contains, and (f) equal between two spatial objects in a two-dimensional space.

4.5 Spatial Data Definition

The data definition in SQL is extended for spatial attributes. The following example shows the definition of the relation *city* with a conventional attribute *name* and a spatial attribute *geometry*.

```
CREATE TABLE city
      ( name            char (20)
      geometry    spatial_2 );
```

In general, a spatial relation will have exactly one spatial attribute which defines the geometry of the object; however, Spatial SQL does not prevent the user from defining several "geometries" for a single object. This possibility might be useful for the representation of an object at different scales and levels of detail. For example, a town on a map may be represented as a polygon in 1:10,000 and as a node at its center of gravity in 1:1,000,000. The data definition in Spatial SQL would allow for a table *city* with two different spatial attributes, like

```
CREATE TABLE city
      ( name                  char (20)
      geometry                spatial-2
      generalizedGeometry     spatial-0 );
```

Spatial attributes in Spatial SQL commands are used equivalently to conventional attributes in SQL either in the SELECT clause as projection, or as a predicate in the WHERE clause.

4.6 Selection by Pointing

Interactive communication with drawings is enabled with the *PICK* qualifier which allows users to formulate queries with reference to spatial objects visible on a screen. PICK is incorporated into the language as a predicate and can qualify each spatial attribute in a WHERE clause. The specification is similar to a value typed from the keyboard; however, the specified value is not a character string but the location of an object identified on the screen.

The semantics of selection by pointing vary depending on the spatial dimension of the target. When pointing to objects rendered in the two-dimensional plane, the target is either the 0 or 1-dimensional object closest to the pointer or the 2-dimensional object which includes the point of selection [Egenhofer 1990b]. Ambiguities in the selection may exist if more than one object of the class identified has the same distance to the pointer position. Such situations may occur when, for instance, a user tries to select a road pointing exactly to the intersection of two roads. Likewise, the selection of an areal object is ambiguous if the user points exactly to

the boundary of two adjacent areas. In such (rare) cases, the user will be offered the possible choices and then asked to identify the target [Egenhofer 1988b].

5 The Graphical Representation Language GRL

The Graphical Representation Language GRL [Egenhofer 1990a] provides tools for the manipulation of the graphical representation of query results. Central to GRL is the concept of the *display environment* which manages the state of the graphical specifications, i.e., how to display query results. During query processing, this information is used to augment the user query with display queries and to render the query result according the display description. Generally, the state of the display environment applies to the *next* Spatial SQL query asked. Unless the user changes the environment with a GRL instruction, the display environment continues to produce a map of the same style. A particular GRL instruction causes the GRL state to be immediately applied to the *current* rendering.

The generic form of a GRL command is

```
<GRL instruction> <graphic specification>;
```

where `<GRL instruction>` specifies the action to take and `<graphic specification>` describes the graphical environment.

GRL distinguishes four instruction types: *set* and *define*, establishing the characteristics of the graphical representation; *cancel*, resetting the environment; and *show*, examining its current values. Environments specified with *set* or *define* are effective for all subsequent Spatial SQL queries and stay valid until they are cancelled or overwritten with a corresponding instruction. *Define* commands persist across sessions, while set commands discontinue with the end of a session. The qualifier *immediately* allows users to update the graphical representation of the current map prior to asking the next query. Both *set* and *define* may be immediately executed.

Six types of graphic specification are distinguished: (1) the display mode, (2) the graphical representation, (3) the scale of the drawing, (4) the window to be shown, (5) the spatial context, and (6) the examination of the content.

5.1 Display MODE

In order to combine several query results in a single drawing, the selection of the appropriate display mode is necessary. In SQL, the SELECT clause is overloaded with the projection of the attributes into the resulting relation and the implied tabular representation. This combination is a major impediment for a query language with more than one kind of representation [Egenhofer 1987]. Spatial SQL solves this problem by the separation of the two issues. The SELECT command performs only the relational projection operation; the query result will be displayed according to the current display type selected with GRL. Display modes are the conventional alphanumeric display and four graphical display types: (1) *new*, starting a new drawing; (2) *overlay*, adding the result onto an existing drawing; (3) *remove*, erasing the result from a drawing; (4) *highlight*, emphasizing the result such that it can be easily recognized.

With the selection of the graphic mode the placement of labels can be directed as well. Non-spatial attributes in the SELECT clause will be represented as labels. Their actual placement remains the task of some name-placement subsystem [Freeman 19871.

5.2 LEGEND

The graphical representation of spatial data has more variety than lexical representation in tabular form. GRL offers the user tools to describe *colors, patterns,* and *symbols* for spatial objects. These graphic attributes can be specified for either an entire relation or specific instances which fulfill the specified constraints. This part of GRL depends upon the user's hardware; therefore, the set of terms is designed to be extensible such that it can be adapted to different environments.

5.3 SCALE and WINDOW

The scale of the graphical representation and the window describing the area to be displayed can be described with the commands SET SCALE and SET WINDOW, respectively. The scale is set by a positive number n, representing a scale factor of I : *n.* The window can be determined either by two pairs of coordinates, two diagonal points selected on a screen drawing, or the minimal bounding rectangle from the result of a Spatial SQL query.

5.4 CONTEXT

The interpretation of graphical representation is extremely dependent on the context and environment in which it is shown. Unlike lexical representation, it is often insufficient to draw only those objects directly requested in the query. The representation of a city as a point in the center of the screen is useless information unless context, such as the boundary of the state, helps to identify its location [Egenhofer 1988b]. GRL allows the user to define spatial relations or specified portions with SET CONTEXT as graphical context which is during query processing merged with the actual user query.

5.5 CONTENT

Because a combination of multiple query results may be shown in a single drawing, a control mechanism is necessary with which the user may examine the content of a drawing. The content is the logical combination of queries the results of which were combined with *Overlay* and *Remove*. In essence, the content gives a single query with which the drawing currently visible could have been produced. Such a single query may be fairly complex. For example, the two following queries may be combined by removing the result of the second from the result of the first.

```
SET MODE new;

SELECT       geometry
FROM         building
WHERE        value > 50.000;

SET MODE     remove;

SELECT       geometry
FROM         building
WHERE        value < 80.000;
```

The content of the drawing is then a map with all buildings with a value greater than or equal to 80.000.

Of course, content is only observable, i.e., users can examine the content with SHOW CONTENT, but they cannot SET or MODIFY it.

6 Query Example

Imagine a geographic database of Penobscot county with towns, parcels, buildings, roads, rivers, and utility lines. An insurance agent wants the following information displayed on a map for a client who intends to buy a home-owner's insurance policy for the building "26 Grove Street" in the town of Orono:

> Show a map of Grove Street in Orono with all buildings, parcel boundaries, and roads. Display the residences in green, commercial buildings in blue, parcel boundaries in black. Cross-hatch roads narrower than 15 ft. Label roads by names.

First, the graphical environment is built describing the various colors, patterns, and symbols used.

```
SET LEGEND
       COLOR        black
       PATTERN      dashed
FOR SELECT          boundary (geometry)
       FROM         parcel;

SET LEGEND
       COLOR        green, blue
FOR SELECT          residence.geometry, commercial.geometry
       FROM         residence building, commercial building
       WHERE        residence.type "Residential" and
                    commercial.type "Commercial";

SET LEGEND
       PATTERN      cross-hatched
FOR SELECT          interior (geometry)
       FROM         road
       WHERE        width < 15;
```

Then the user identifies the window of interest and sets a context for the roads.

```
SET    WINDOW
       SELECT geometry
       FROM   road
       WHERE  town.name = "Orono";

SET    CONTEXT
FOR    road.geometry
       SELECT  parcel.geometry, building.geometry, road.name
       FROM    road, parcel, building;
```

Finally, the user selects to draw a new map and enters the actual user query, a brief Spatial SQL statement.

```
SET    MODE new;

SELECT              road.geometry
FROM                road, town
WHERE               town.name = "Orono" and
                    road.name = "Grove Street" and
                    road.geometry INSIDE town.geometry;
```

The result is the desired map (Fig. 3).

With further Spatial SQL commands the user requests more information about the objects displayed or manipulates the graphical properties of the rendering. For example, the following commands will help the insurance agent to identify the building with address "26 Grove Street":

```
SET MODE       highlight;

SELECT         building.geometry
FROM           building
WHERE          address = "26 Grove Street";
```

The insurance agent’s primary interest is in fire fighting. How far is the building from the next fire station?

```
SET MODE      alpha;

SELECT        distance (building.geometry, firestation.geometry), firestation.address
FROM          building, building firestation
WHERE         building = PICK and
              firestation.type = "Fire Station";
```

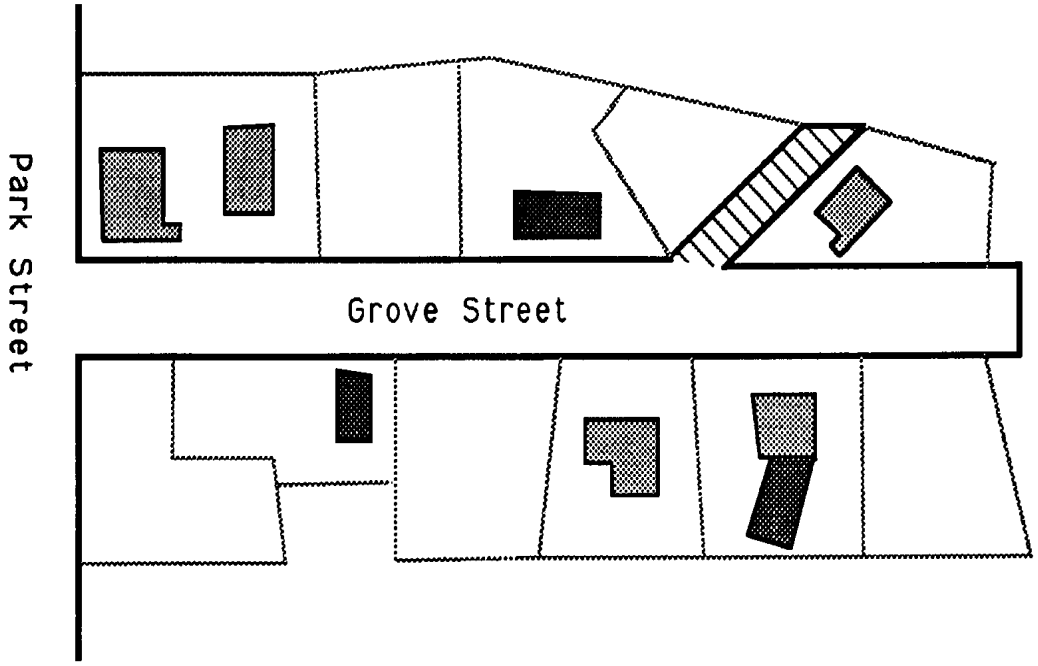


Figure 3: Map of Grove Street showing all buildings, parcel boundaries, and roads.

```
WHERE       building = pick and
            firestation.type = "Fire Station";
```

The result is a list of fire stations and their distances to the building which the user selected by pointing (PICK). Note that the previously defined window restricts the fire stations to be located in Orono.

To check the accessibility to a water hydrant from "26 Grove Street" the insurance agent overlays the water hydrants, located within 100 ft of the building, over the current map.

```
SET LEGEND
      COLOR        red
      SYMBOL       "2mm disk"
FOR   SELECT       geometry
      FROM         utility
      WHERE        type = "Water Hydrant";

SET MODE overlay;

SELECT geometry
FROM   utility, building
WHERE  type = "Water Hydrant" and
       building.geometry = PICK and
       distance (building.geometry, utility.geometry) < 100;
```

Are there any parcels on Grove Street which had fire accidents since 1980? If yes, show them on the map.

```
SET-IMMEDIATELY COLOR red
FOR-SELECT          interior (geometry)
      FROM         parcel
      WHERE  fire-damage-date >= 1980;
```

The insurance agent checks the values of the buildings on the parcels adjacent to "26 Grove Street."

7 Conclusion

The functionality of SQL has been extended with spatial properties to fulfill eight essential requirements for a spatial query language. Spatial SQL is characterized by minimal extensions to SQL and the introduction of the Graphical Representation Language (GRL). The eight requirements for a spatial query language are satisfied with: (1) the introduction of a spatial data type and the corresponding operations and relationships; (2) the graphical representation directed from GRL; (3) the display MODES New, Overlay, Remove, Diff, and *Highlight* in GRL to combine query results into a single drawing; (4) the definition of CONTEXT in GRL; (5) the examination of CONTENT from GRL; (6) selection by pointing via the PICK qualifier in WHERE clauses; (7) the manipulation of the graphical representations of objects with colors, patterns, and symbols; (8) the examination of the map LEGEND in GRL; (9) the combination of both graphical and alphanumeric data in a single result which enable objects to be labeled- (10) graphical representations in specific map scales; and (11) the selection of a spatial query window to which queries will refer.

Several features of Spatial SQL have an object-oriented flavor, such as the complex abstract data type spatial and its subtypes for different spatial dimensions. Recently, object-oriented SQL versions have been proposed as general ad hoc query languages for object-oriented databases, e.g., OSQL for IRIS [Fishman 1986], HDBL [Linnemann 1988], and the 02 query language [Bancillion 1989]. HDBL and OSQL allow users to define complex data types, including generalization hierarchies, and corresponding operations; however, the extension of a database language with a new abstract data type [Date 1988] [Ong 1984] [Stonebraker 1983] is only one of the requirements for a spatial query language. All object-oriented SQL extensions have not made any provisions for the issues related to the graphical representation of query results.

The implementation of Spatial SQL has reached the following state: a parser was implemented for both Spatial SQL and GRL commands. Parsed Spatial SQL commands are translated into Horn clauses for query optimization and distribution among a relational database management system and a spatial file system [Hudson 1989]. Optimization takes into account spatial indexing, similar to strategies in other spatial databases [Giiting 1989] [Ooi 1989b]. A prototype for this component exists as well. It is planned to translate GRL commands into Horn clauses and to merge them with the actual user query, such that both requests can be processed in a single stream. Query results to be graphically displayed will be loaded into the spatial DBMS which provides graphical output.

Spatial SQL and GRL commands are verbose if a user has to type them from a keyboard. Considerable increase of the usability of this spatial query language is expected by incorporating the command line structure into a human interface, as proposed in [Egenhofer 11988b]. The incorporation of the direct-manipulation device into the query language is a first step toward an improved interaction with maps using pointing instead of typing. Operations like pan and zoom are good candidates to exploit this concept for which appropriate metaphors are being examined [Jackson 1990]. Likewise, the implementation of GRL as a visual language is likely to boost its usability. For example, instead of typing a name for a color or pattern, users may make their choice from a control panel which presents a series of colors and patterns, and also allows the users to create new ones. A cartographic editor to create and manipulate cartographic symbols is an essential component of such a user interface [Steiner 1989]. Finally, the visualization of prototypical spatial relations may reduce the time for learning a specific terminology and help users in the selection of the correct configuration.

Acknowledgement

Andrew Frank's expertise and advice were helpful contributions to this paper. Thanks also to Renato Barrera and Bruce Palmer for many stimulating discussions, Doug Hudson who made many helpful comments on the language design, and Robert Cicogna who helped with the preparation of this paper.

References

[ANSI 19861 X3H2-86-2 American National Standard Database Language SQL. American National Standards Institute, January 1986.

[Ariav 1986] G. Ariav. A Temporally Oriented Data Model. ACM Transactions on Database Systems, 11(4):499-527, December 1986.

[Astrahan 1976] M. Astrahan, M. Blasgen, D. Chamberlin, P. Eswaran, J. Gray, P. Griffiths, W. King, R. Lorie, P. McJones, J. Melil, G. Putzolu, 1. Traiger, B. Wade, and V. Watson. System R: A Relational Approach to Data Base Management. ACM Transactions on Database Systems, 1(2):97-137, June 1976.

[Bancilhon 1989] F. Bancilhon, S. Cluet, and C. Delobel. A Query Language for the 02 Object-Oriented Database System. In: Second International Workshop on Database Programming Languages, pages 122-138, Gleneden Beach, OR, June 1989.

[Barrera 1981] R. Barrera and A. Buclimann. Schema Definition and Query Language for a Geographical Database System. IEEE Transactions on Computer Architecture: Pattern Analysis and Image Database Management, 11:250-256, 1981.

[Berman 1977] R. Berman and M. Stonebraker. GEO-QUEL, A System for the Manipulation and Display of Geographic Data. ACM Computer Graphics, 11(2):186-191, 1977.

[Bertin 1983] J. Bertin. Semiology of Graphics. The University of Wisconsin Press, Madison, WI, 1983.

[Burton 1979] W. Burton. Logical and Physical Data Types in Geographic Information Systems. Geo-Processing, 1(2):167-181, 1979.

[Chamberlin 1976] D. Chamberlin, M. Astrahan, K. Eswaran, P. Griffiths, R. Lorie, J. Mehl, P. Reisner, and B. Wade. SEQUEL 2: A Unified Approach to Data Definition, Manipulation, and Control. IBM Journal of Research and Development, 20(6):560-575, November 1976.

[Chang 1980] N.S. Chang and K.S. Fu. Query-by-Pictorial-Example. IEEE Transactions on Software Engineering, SE-6(6):519-524, November 1980.

[Chang 1981] S.K. Chang and L.K. Kunii. Pictorial Database Systems. Computer, 15(11):13-21, November 1981.

[Codd] 9881 E.F. Codd. Fatal Flaws in SQL. Datarnation, 34(16), August 1988.

[Cox 1979] N. Cox, B. Alfred, and.D. Rhind. A Relational Data Base System and a Proposal for a Geographic Data Type. Geo-Processing, 1(3):217-229, 1979.

[Date 1988] C.J. Date. Defining Data Types in a Database Language. SIGMOD Record, 17(2):53-76, June 1988.

[Egenhofer 1987] M. Egenhofer. An Extended SQL Syntax To Treat Spatial Objects. In: Y.C. Lee, editor, Second International Seminar on Trends and Concerns of Spatial Sciences, pages 83-95, Fredericton, New Brunswick, Canada, 1987.

[Egenhofer 1988a] M. Egenhofer. Graphical Representation of Spatial Objects: An ObjectOriented View. Technical Report 83, Department of Surveying Engineering, University of Maine, Orono, ME, July 1988.

[Egenhofer 1988b] M. Egenhofer and A. Frank. Towards a Spatial Query Language: User Interface Considerations. In: D. DeWitt and F. Bancilhon, editors, 14th International Conference on Very Large Data Bases, pages 124-133, Los Angeles, CA, August 1988.

[Egenhofer 1989a] M. Egenhofer. A Formal Definition of Binary Topological Relationships. In: W. Litwin and H.-J. Schek, editors, Third International Conference on Foundations of Data Organization and Algorithms (FODO), Paris, France, pages 457-472, Springer-Verlag, New York, NY, June 1989.

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

[Egenhofer 1990a] M. Egenhofer. Extending SQL for Cartographic Display. Cartography and Geographic Information Systems, 1990.

[Egenhofer 1990b] M. Egenhofer. Manipulating the Graphical Representation of Query Results in Geographic Information Systems. In: Workshop on Visual Languages, pages 119-124, Skokie, IL, October 1990.

[Egenhofer 1990c] M. Egenhofer and J. Herring. A Mathematical Framework for the Definition of Topological Relationships. In: K. Brassel and H. Kishimoto, editors, Fourth International Symposium on Spatial Data Handling, pages 814-819, Zurich, Switzerland, July 1990.

[Egenhofer 1990d] M. Egenhofer and R. Franzosa. Point-Set Topological Spatial Relations. submitted for publication, 1990.

[Ehrich 1988] H.-D. Ehrich, F. Lohmann, K. Neumann, and 1. Ramm. A Database Language for Scientific Map Data. Geologisches Jahrbuch, A(104): 139-152,1988.

[Embley 1986] D.W. Embley and G. Nagy. Toward A High-Level Integrated Image Database System. Technical Report, Image Processing Laboratory, Rensselaer Polytechnic Institute, Troy, NY, April 1986.

[Fishman 1986] Fishman, D. Beech, H. Cate, E. Chow, T. Connors, J. Davis, N. Derrett, C. Hoch, W. Kent, P. Lyngbaek, B. Mahbod, N. Neimat, T. Ryan, and M. Shan. Iris: An Object-Oriented Database Management System. ACM Transactions on Office Information Systems, 5(l):48-69, January 1986.

[Frank 1981] A. Frank. Applications of DBMS to Land Information Systems. In: C. Zaniolo and C. Delobel, editors, Seventh International Conference on Very Large Data Bases, pages 448-453, Cannes, France, 1981.

[Frank 1982] A. Frank. MAPQUERY-Database Query Language for Retrieval of Geometric Data and its Graphical Representation. ACM Computer Graphics, 16(3):199207, July 1982.

[Frank 1984a] A. Frank. Computer Assisted Cartography-Graphics or Geometry. Journal of Surveying Engineering, 110(2):159-168, August 1984.

[Frank 1984b] A. Frank. Requirements for Database Systems Suitable to Manage Large Spatial Databases. In: International Symposium on Spatial Data Handling, Zurich, Switzerland, August 1984.

[Freeman 1987] H. Freeman and J. Alin. On the Problem of Placing Names in a Geographic Map. International Journal of Pattern Recognition and Artificial Intelligence, l(l):121-140, 1987.

[Friedell 1982] M. Friedell, J. Barnett, and D. Kramlich. Context-Sensitive, Graphic Presentation of Information. ACM Computer Graphics, 16(3):181-188, July 1982.

[Go 1975] A. Go, M. Stonebraker, and C. Williams. An Approach to Implementing a Geo-Data System. Technical Report, Memo ERL-M529, Electronics Research Laboratory, University of California, Berkeley, CA, June 1975.

[Guting 1988] R. Gilting. Geo-Relational Algebra: A Model and Query Language for Geometric Database Systems. In: J. Schmidt, S. Ceri, and M. Missikoff, editors, Advances in Database Technology-EDBT '88, International Conference on Extending Database Technology, Venice, Italy, pages 506-527, Springer Verlag, New York, NY, 1988.

[Guting 1989] R. Gilting. Gral: An Extensible Relational Database System for Geometric Applications. In: P. Apers and G. Wiederhold, editors, Fifthteenth International Conference on Very Large Data Bases, pages 33-44, Amsterdam, August 1989.

[Herot 1980] C. Herot. Spatial Management of Data. ACM Transactions on Database Systems, 5(4):493-513, December 1980.

[Herring 1988] J. Herring, R. Larsen, and J. Shivakumar. Extensions to the SQL Language to Support Spatial Analysis in a Topological Data Base. In: GIS/LIS '88, pages 741-750, San Antonio, TX, November 1988.

[Hudson 1988] D. Hudson. Combined Spatial/11 and RDB Database Operation: Query Coordination and Optimization Strategies. 1988. Internal Documentation, University of Maine, Orono, Department of Surveying Engineering, Orono, ME.

[Hudson 1989] D. Hudson. A Unifying Database Formalism. In: ASPRS/ACSM Annual Convention, pages 146-153, Baltimore, MD, April 1989.

[Ingram 1987] K. Ingram and W. Phillips. Geographic Information Processing Using a SQL-Based Query Language. In: N.R. Chrisman, editor, AUTO-CARTO 8, Eighth International Symposium on Computer-Assisted Cartography, pages 326335, Baltimore, MD, March 1987.

[Jackson 1990] J. Jackson. Developing an Effective Human Interface for Geographic Information Systems Using Metaphors. In: ACSM-ASPRS Annual Convention, pages 117-125, Denver, CO, March 1990.

[Kainz 1990] W. Kainz. Spatial Relationships-Topology versus Order. In: K. Brassel and H. Kishimoto, editors, Fourth International Symposium on Spatial Data nandling, pages 814-819, Zurich, Switzerland, July 1990.

[Keating 1987] T. Keating, W. Phillips, and K. Ingram. An Integrated Topologic Database Design for Geographic Information Systems. Photogrammetric Engineering & Remote Sensing, 53(10):1399-1402, October 1987.

[Lee 1983] Y.C. Lee and K.S. Fu. A CSG Based DBMS for CAD/CAM and its Supporting Query Language. In: Engineering Design Applications, Annual Meeting, Database Week, pages 123-130, San Jose, CA, May 1983.

[Linnemann 1988) V. Linnemann, K. Kdspert, P. Dadam, P. Pistor, R. Erbe, A. Kemper, N. Siidkamp, G. Walch, and M. Wallrath. Design and Implementation of an Extensible Database Management System Supporting User Defined Data Types and Functions. In: D. DeWitt and F. Bancilhon, editors, 14th International Conference on Very Large Data Bases, pages 294-305, Los Angeles, CA, August 1988.

[Lohman 1983] G. Lohman, J. Stoltzfus, A. Benson, M. Martin, and A. Cardenas. Remotely-Sensed Geophysical Databases: Experience and Implications for Generalized DBMS. In: D. DeWitt and G. Gardarin, editors, SIGMOD 83, Annual Meeting, pages 146-160, San Jose, CA, 1983.

[Lorie 19881 R. Lorie and H.-J. Schek. On Dynamically Defined Complex Objects and SQL. In: K.R. Dittrich, editor, Advances in Object-Oriented Database SystemsProceedings of the 2nd International Workshop on Object-Oriented Database Systems, Bad Miinster am Stein-Ebernburg, F.R. Germany, pages 323-328, SpringerVerlag, New York, NY, September 1988.

[Luk 1986] W.S. Luk and S. Kloster. ELFS: English Language From SQL. ACM Transactions on Database Systems, 11(4):447-472, December 1986.

[McKeown 1983] D. McKeown. MAPS: The Organization of a Spatial Database System Using Imagery, Terrain, and Map Data. Technical Report CMU-CS-83-1.36, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, July 1983.

[McLeod 1983] D. McLeod, K. Narayanaswamy, and K.V. Bapa Rao. An Approach to Information Management for CAD/VLSI Applications. In: Engineering Design Applications, Annual Meeting, Database Week, pages 39-50, San Jose, CA, May 1983.

[Ong 1984] J. Ong, D. Fogg, and M. Stonebraker. Implementation of Data Abstraction in the Relational Database System INGRES. SIGMOD Record, 14(l), March 1984.

[Ooi 1989a] B.C. Ooi, R. Sacks-Davis, and K. McDonell. Extending a DBMS for Geographic Applications. In: IEEE Fifth International Conference on Data Engineering, pages 590-597, Los Angeles, CA, February 1989.

[Ooi 1989b] B. Ooi and R. Sacks-Davis. Query Optimization in an Extended DIB,MS. In: W. Litwin and H.-J. Schek, editors, Third International Conference on Foundations of Data Organization and Algorithms (FODO), Paris, France, pages 48-63, Springer-Verlag, New York, NY, June 1989.

[Roussopoulos 1985] N. Roussopoulos and D. Lefflier. Direct Spatial Search on Pictorial Databases Using Packed R-Trees. In: International Conference on Management of Data, SIGMOD RECORD, pages 17-31, 1985.

[Roussopoulos 1988] N. Roussopoulos, C. Faloutsos, and T. Sellis. An Efficient Pictorial Database System for PSQL. IEEE Transactions on Software Engineering, 14(5):630-638, May 1988.

[Sarda 1990] N. Sarda. Extensions to SQL for Historical Databases. IEEE Transactions on Knowledge and Data Engineering, 2(2):220-230, June 1990.

[Sikeler 1985] A. Sikeler. Examination of Storage Structures for 3-Dimensional Objects (in German). Technical Report, University Kaiserslautern, 1985.

[Snodgrass 1987] R. Snodgrass. The temporal query language TQUEL. ACM Transactions on Database Systems, 12(6):247-298, June 1987.

[Steiner 1989] D. Steiner, M. Egenhofer, and A. Frank. An Object-Oriented Carto-Graphic Output Package. In: ASPRS-ACSM Annual Convention, pages 104-113, Baltimore, MD, March 1989.

[Stonebraker 1976] M. Stonebraker, E. Wong, and P. Kreps. The Design and Implementation of INGRES. ACM Transactions on Database Systems, 1(3):189-222, September 1976.

[Stonebraker 1983] M. Stonebraker, B. Rubenstein, and A. Guttman. Application of Abstract Data Types and Abstract Indices to CAD Databases. In: D. DeWitt and G. Gardarin, editors, ACM SIGMOD Conference on Engineering Design Applications, pages 107-113, San Jose, CA, 1983.

[Tomlin 1990] C.D. Tomlin. Geographic Information Systems and Cartographic Modeling. Prentice-Hall, Englewood Cliffs, NJ, 1990.

[von Biiltzingsloewen 1987) G. von Biiltzingsloewen. Translating and Optimizing SQL Queries Having Aggregates. In: P. Stocker and W. Kent, editors, 13th International Conference on Very Large Data Bases, pages 235-243, Brighton, England, September 1987.

[Westlake 1990] A. Westlake and I. Kleinsclimidt. The Implementation of Area and Membership Retrievals in Point Geography using SQL. In: Z. Michalewicz, editor, Statistical and Scientific Database Management, Fifth International Conference, V SSDBM, Charlotte, NC, pages 200-218, Springer-Verlag, New York, NY, April 1990.

[Zloof 1977] M.M. Zloof. Query-by-Exarnple: A Database Language. IBM Systems Journal, 16(4):324-343, 1977.

LOBSTER: Combining AI and Database Techniques for GIS*

Max J. Egenhofer
Andrew U. Frank
National Center for Geographic Information and Analysis
and
Department of Surveying Engineering
University of Maine
Orono, ME 04469, USA
MAX@MECAN1.bitnet
FRANK@MECAN1.bitnet

Abstract

The powerful logic-based concept of Prolog has been integrated with a database suitable for spatial data handling to form a database query language that is more flexible and powerful than the currently used SQL. This experimental implementation, called LOBSTER, allowed researchers to explore a number of areas of a GIS. Examples from object-oriented modeling, geomorphology, and query optimization show the application of such a language. Problems encountered during the application of LOBSTER include the absence of consistency checking during input of rules and facts, and the lack of appropriate techniques to detect cyclic rule definitions. Nevertheless, the experimental implementation showed that these techniques were extremely valuable for GIS.

1 Introduction

To develop a flexible and powerful program system for geographic information systems (GIS) is a challenging task. It is particularly difficult to construct programs that can assist users for all the different functions they expect from a GIS [Frank 1984] [Smith 1987]. During the past decade, advanced methods and techniques from computer science have been integrated into GIS. The use of high-level programming languages is commonplace today and the designers of GIS software use modern software engineering techniques [Aronson 1983]. Database management systems and their principles have been applied and the specific requirements of spatial data handling studied [Frank 1988]. Finally, it is recognized that some methods from artificial intelligence (AI) could be beneficial for GIS [Abler 1987] [Peuquet 1987] [Robinson 1987b] [McKeown 1987].

Computer systems are essentially formal systems that manipulate symbols according to formal rules. These systems do not understand-in the sense that human beings' understand-the meanings of the symbols or what they stand for. They follow the instructions of their programs with blinding speed, but without any "common sense." Computer systems treat *formal models* which consist of two parts: (1) a *theory* with a collection of expressions in a formal language and (2) an agreed-upon *interpretation* of formal expressions which link the symbols used in the formal system with reality. The derivation of information is the process of proving a specific proposition within such a theory. The best-known language for a formal model is *first-order logic* which expresses *facts* and *rules* in a single, formalized matter [Gallaire 1984b] and derives knowledge by using formal rules.

The deductive power of logic inference systems is typically used in AI systems (Barr 1982] [Hayes-Roth 1983]. Geographic Information Systems need these methods to help integrate data from different sources into a unified system [Robinson 1987a]. A deficiency of any AI-based system is the quantitative difference between AI/expert systems and database management systems [Mylopoulos 1981]: while database management systems are good for the storage of large amounts of data elements (records) from a very few types (structured data), Al systems store a smaller number of facts, but of a much larger variety of types (unstructured data). In this paper, methods from Al research are combined with database management techniques to make both available to GIS. A particular system has been implemented which allowed us to conduct a number of experiments in promising areas for the use of Al in GIS.

The remainder of this paper is organized as follows: the next section discusses the need for intelligent GIS query languages. Prolog, an Al programming language, is proposed as a powerful query language if integrated with a database management system.

* This work was partially funded by a grants from NSF under grant number IST 86-09123 and Digital Equipment Corporation. The support from NSF for the NCGIA under grant number SES 88-10917 is gratefully acknowledged.

LOBSTER is such a persistent language combining concepts of the Prolog programming language with database management techniques. The integration of DBMS and Al language are discussed as well as the implementation of the inference machine. The last section reports on some of our GIS test applications using LOBSTER, such as the implementation of object-oriented abstraction mechanisms, feature extraction in geomorphology, and query optimization in a distributed database environment. The paper concludes with a summary of drawbacks encountered during the use of LOBSTER as a Prolog-based query language.

2 GIS Query Languages

The production of spatial information on demand is the motivation for spatial query languages. A *query language* is a general means to request information about the contents of a database. Users formulate their requests to the database by describing their needs ("What to retrieve") and the desired representation of the result ("How to represent the results")*A spatial query language* Is a tool suitable to interrogate spatial databases.

2.1 Database Query Languages

Database query languages are tools to facilitate access to a database and have been investigated by computer scientists for more than a decade. The term *query* refers to a statement requesting data to be retrieved from a database. Query languages are best-known with respect to (relational) databases. SQL, an acronym for Structured Query Language [Chamberlin 1976], is the standard relational query language [ANSI 1986] and enjoys popularity in traditional database applications, such as accounting. Based on the underlying relational data model [Codd 1970], SQL deals exclusively with relations, combinations of relations, and some "syntactic sugar" added to relational algebra, such as arithmetic capabilities, assignment of results to relations, and aggregate functions. Although SQL is very popular and has been standardized, there has been criticism that SQL queries can be difficult to understand [Luk 1986] and are particularly cumbersome to use for complex engineering applications.

The fundamental structure of SQL is the SELECT-FROM-WHERE block. The SELECT clause determines the attributes to display; the FROM clause describes the data sets needed to solve the query; and the optional WHERE clause specifies constraints upon the items to be retrieved. For example, the request for all lines with start or end nodes within a box described by two pairs of x- and y-values is formulated in SQL as follows:

```
SELECT      line.id
FROM        line, node
WHERE       20000 < x and x < 30000 and
            25000 < y and y < 30000 and
            (line.start = node.id or line.end = node.id);
```

Asked against a database containing the relations **node** and **line** (figure 1), the result is a relation with the two tuples B and C which are automatically displayed on the screen in a tabular format.

point	id	x	y
	2	22399.28	22379.72
	3	23874.39	25479.93
	8	19829.83	29878.98

line	id	start	end
	A	2	3
	B	3	8
	C	8	2

Figure 1: The data sets *point* with the attributes *id*, *x*, and *y*; and *line* with *id*, *start*, and *end*.

Quel, the query language for the Ingres database management system [Stonebraker 1976], closely imitates the tuple relational calculus [Codd 1972] and has the same expressive power as SQL, i.e., any query asked in SQL can be also asked in Quel.

The third major query language is Query-by-Example [Zloof 1977]. It supports users with *skeleton tables* to be filled out like forms making the language more user friendly and easier to learn [Reisner 1981] than conventional one- dimensional languages as command strings. Some additional conventions are used, e.g., an underscore character precedes domain variables to distinguish them from constants. Figure 2 shows the same query as above in Query-by-Example.

line	id	start	end
	P._n	_x	
	P._n		_x

point	id	x	y
	_x	> 20,000 and < 30,000	> 25,000 and < 30,000

Figure 2: A Query-by-Example instruction to print (P.) the lines starting or ending inside of the rectangle (20,000 < x < 30,000, 25,000 < y < 30,000).

2.2 Requirements for GIS Query Languages

GIS applications place specific demands on the expressive power and capacity of their query languages. Conventional query languages can certainly be used to access spatial objects stored in databases; however, it is difficult for them to express queries which involve particular spatial properties [Frank 1982a] [Egenhofer 1988] [Laurini 1989]. The following examples demonstrate typical GIS queries and underscore the problems traditional (relational) query languages have with their formulation and processing.

Frequently, GIS users ask for quantitative spatial information, such as the distance between two objects. Traditional query languages lack geometric concepts and do not support the formulation of user queries with spatial terms. Users with limited mathematical skills have difficulties in handling such a system. For example, to retrieve not only the lines starting or ending in a box, but also those crossing through it, users must explicitly formulate complex equations for line intersections. The requirement of such detailed mathematical knowledge makes "pure" SQL too complex to use for spatial applications.

Another complex query in the context of a GIS is to find the largest connected forest area which contains a specific parcel. This request for the *transitive closure* translates into the two operations: (1) to find the parcel *X* and place it in a set *S* and (2) to repeat-until the set *S* does not grow anymore-the operation: for each parcel *P* in *S* find all its neighbors *N*, and if the parcel type is forest then add *N* to the set *S*. Traditional database query languages lack the concepts of loops and recursion necessary to solve such queries, and therefore, cannot be used to formulate such queries.

Other GIS query language requirements include the graphical representation of query results and the display of context to make certain queries understandable [Egenhofer 1989c]

3 Prolog and Database Management Systems

3.1 Prolog

A Prolog-like language may be used as a query language to a GIS based on a database management system that can deal with large numbers of (spatial) data records. Prolog [Clocksin 1981] is an implementation of a subset of first-order predicate logic [Gallaire 1984b] It is based on *facts* and *rules* which are expressed as Horn clauses. A *clause* is a canonical representation of *predicates* $a_o \ldots a_n$, $b_o \ldots b_m$ in the form

$$a_0 \text{ OR } a_1 \text{ OR } \cdots \text{ OR } a_n \text{ IF } b_0 \text{ AND } b_1 \text{ AND } \cdots \text{ AND } b_m. \qquad (1)$$

The left hand side of the clause, called the *consequent*, is the combination of all disjunctions (ORs) and the right hand side, the *antecedent*, has all conjunctions (ANDs). In a *Horn clause* the consequent predicates, i.e. the a_i's, are restricted to zero or one instance, that is

$$a_0 \text{ IF } b_0 \text{ AND } b_1 \text{ AND } \cdots \text{ AND } b_m. \qquad (2)$$

The following example demonstrates the use a Prolog language based on the set of points and lines in figure 1. Predicates tagged by asterisks denote the definition of clauses, while untagged clauses stand for queries, and constants are capitalized, whereas variables start with lower case.

```
*point (2, 22399.28, 22379.72).
*point (3, 23874.39, 25479.93).
*point (8, 19829.83, 29878.98).

*line (A, 2, 3).
*line (B, 3, 8).
*line (C, 8, 2).
```

Given the implementation of the predicate inBox (x, y, xLow, xHigh, yLow, yHigh), the following clauses define the rules for the inclusion of start and end point within a box:

```
*linePoints (1, p) IF line (1, p, end).
*linePoints (1, p) IF line (1, start, p).
*lineInBox (1, xl, yl, xh, yh) IF linePoints (1, p), node (p, X, Y), inBox (x, y, xl, yl, xh,
yh).
```

Prolog's inference mechanism allows then for the derivation of the query result:

```
lineInBox (lineId, 20000, 25000, 30000, 30000).

lineId = B
lineId = C
```

3.2 Combining Prolog with a Database Management System

A Prolog system is often viewed as a programming language, but it also contains certain aspects of a database management system [Kowalski 1979], such as storage and retrieval of data and information. The use of Prolog or similar logic programming methods for database management have been proposed [Gallaire 1984a]. Using a Prolog system as a database [Motro 1984] allows users to store unstructured-or minimally structured-facts without being aware of a database schema. Data and metadata are stored in the same format, so that users need not distinguish between and can search them the same way. Another approach, coupling an existing database management system with a separate Prolog system [Vassilou 1983], makes the potential of the Prolog programming language available for user interaction in an interactive environment with existing, traditionally structured databases Parke 1984]. In this combination, the database system stores the structured data, while the Prolog system is used as an expert system or a tool for an enhanced user interface. Such interfaces to database management systems have been recently integrated into some commercial Prolog systems. Specific attention has to be paid to query processing. Performance will seriously degrade if the inference engine frequently passes control and data from the Prolog system to the database and vice-versa to process predicates one instance at a time.

3.3 Persistent Programming Languages and Prolog

The extension of a programming language with database management capacities is frequently referred to as *a persistent programming language*. Persistent programming languages have been designed and implemented as extensions of object-oriented programming languages, such as Smalltalk [Goldberg 1983] and C++ [Stroustrup 1986], but lack the simplicity and inference power of a Prolog language.

Standard Prolog [Clocksin 1981] leaves the provision for long term storage of facts and rules to file storage. Hence, we combined Prolog with a database management system to construct a *persistent Prolog*. Users can store data, structured according to the database schema, with Prolog facts and rules in the same database representing unstructured data. Simultaneously, they can use the inference mechanism to exploit the data.

Generally, most database management systems based on the network or relational data model can be used to support an inference mechanism of the form described. A database management system then serves as a general storage and retrieval system for clauses. This replaces the particular systems built in present Prolog implementations. The major extension is the use of disk storage and access methods; however, for GIS applications, the database system must respond to a number of specific requirements [Frank 1988], for example:

- object-oriented database design (Dittrich 1986],
- generalization/specialization as abstraction methods [Borgida 1984],

- suitability for modeling of geometric data [Harder 1985] with high-level abstractions of geometric objects, operations, and classes [Egenhofer 1988] [GiAing 1988],

- fast access based on spatial location [Frank 1981].

The change in the environment-database in lieu of programming-aggravates some of the well-'town problems of Prolog:

- User input of new facts and rules must be checked for consistency, e.g., comparing the spelling of new facts against previously stored ones.

- Execution speed with large spatial data collections must be improved so that acceptable response times can be guaranteed.

4 LOBSTER

LOBSTER is a persistent Prolog interpreter [Frank 1984] using the PANDA database management system [Frank 1982b]. PANDA incorporates many object-oriented concepts, such as generalization and association, extensibility with user-defined abstract data types, and spatial storage and access methods [Egenhofer 1989b].

LOBSTER can be distinguished from the standard Prolog implementation [Clocksin 1981] in several aspects:

- Persistency of rules and facts: The rules and facts users store are kept on disk in a permanent database and available for any future work. In contrast, standard Prolog demands that the used rules and facts are loaded into main memory at the beginning of each session.

- Organization of rules and facts into groups: The persistency of all facts and rules requires that users have some tools to organize them so that they can keep track of what they had previously defined.

- Extensibility: New built-in predicates, written in a conventional programming languages, such as Pascal, can be easily implemented and integrated into the LOBSTER environment so that their actual implementation is hidden from the users.

Newer commercial Prolog products do provide some similar features, including access to relational database management systems.

Central to LOBSTER is the combination of a Prolog interface with a database management system to allow users to store GIS data and use the Prolog language and interpreter for building a query language. These two systems must be linked so that the Prolog interpreter has access to the data stored in the database management system and that these data may appear as facts in the Prolog system. The link between the two systems is achieved in two steps:

- Operations for database access are coded and integrated into the Prolog interpreter such that they appear as regular Prolog predicates, so-called *built-ins*, providing a low-level access to database facts from Prolog.

- Mappings are defined from the conceptual database schema to Prolog predicates and then implemented as Prolog rules using the built-ins for database access.

In order to store facts in a database, a database schema had to be designed. Figure 3 shows a solution in an extended enti ty-relationship diagram.

The following example demonstrates how a Prolog rule is stored in the database according to this structure. The rule
is stored as a *clause* with two *predicates* (`grandFather, father`) and three *symbols* (`x, y, xy`). The clause consists of three *atomic formulae* (`grandFather (x, y)` as the *consequent* atom, `father (x, xy)` and `father (xy, Y)` as the *antecedent* atomic formulae). For each atomic formula, the corresponding variables are recorded with their number in the clause (e.g., x=I, y=2 for `grandFather`). In the clause, each variable is connected witb the respective symbol, either as *const* for a constant or *bound/unbound* for a variable before and after binding it to a value, respectively.

The implementation of LOBSTER follows the Prolog method of a *depth first search* and uses the facts in the order they are encountered in the linkage of predicates, atoms, and consequent clause. For use as a database retrieval system, the interpreter has to return with each success so the application can use the data in any way it is necessary, i.e., the Prolog interpreter works as though it

was a co-routine. This excludes a simple recursive implementation of the interpreter and requires explicit storage of the state of the interpreter during query processing to continue the search with backtracking after a solution has been found and processed. The backtracking algorithm is inherently sequential, using one data element at a time and its formulation in a navigational data manipulation language makes no problems. It cannot easily take advantage of the set oriented interface of a relational database. In order to reduce the number of physical disk accesses necessary for each step, physical clustering of records, as provided by PANDA, is beneficial.

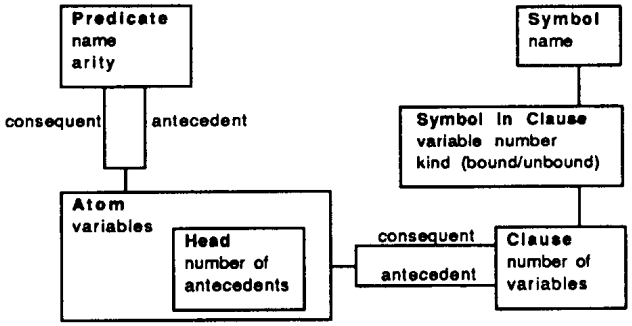


Figure 3: Database schema of LOBSTER in an extended Entity-Relationship diagram.

```
grandFather (x, y) IF father (x, xy) and father (xy, y)
```

5 Applications

In this section, some experimental. applications will be presented which were built upon LOBSTER exploiting the power of logic programming, the database approach, and the extensibility.

5.1 Object-Oriented Abstraction Mechanisms

Object-oriented modeling is an innovative approach to designing software systems for complex situations in dealing with real-world problems as they occur in GIS [Dittrich 1986]. It pursues the integration of traditionally separated methods used in DBMS, programming languages, and Al [Mylopoulos 1981] [Brodie 1984] and employs powerful abstraction mechanisms, such as *generalization* [Borgida 1984], *association* [Brodie 1981], and *aggregation* [Smith 1977]. The concepts of *inheritance* [Goldberg 1983] [Cardelli 1984] and *propagalion* [Rumbaugh 1988], originating from programming languages, play an important role for GIS modeling [Egenhofer 1989a]. A system like LOBSTER is particularly well-suited to demonstrate these sophisticated abstraction mechanisms in a concise fashion. Data and metadata are described in a uniform way so that users may easily exploit metadata for the formulation of rules and queries.

Inheritance is a method of defining a class in terms of one or more other, more general classes [Dahl 1966], called an *is_a* hierarchy [Mylopoulos 1984]. Properties common for a superclass and its subclasses are defined only once with the superclass-and inherited by all objects in the subclass. Subclasses may have additional, specific properties and operations which are not shared by the superclass. Figure 4 shows a generalization hierarchy with three levels of classes. The properties of a *building,* such as *address* and *owner,* are inherited to the subclass *residence,* and also transitively to the sub-subclasses *rural residence* and *urban residence.*

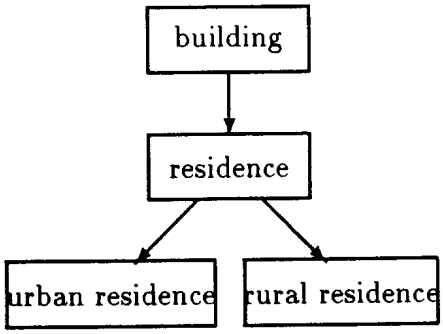


Figure 4: Properties are transitively inherited from a superclass to all its subclasses, the sub-subclasses, etc.

LOBSTER has been used to prototype these concepts so that experiments could be run in a GIS environment [Egenhofer 1989a]. Each property of a class is expressed as a predicate of the form `p (class, property).` Generalization is described as the *is_a*-predicate of the form `is_a (subclass, superclass).` The following facts describe the model depicted in figure 4.

```
*p (Building, Address).
*p (Building, Owner).
*p (Residence, Resident).

*is_a (RuralResidence, Residence).
*is_a (UrbanResidence, Residence).
*is_a (Residence, Building).
```

Inheritance is then defined by the predicate `properties` which recursively derives the properties associated with a class and all its superclasses.

```
*properties (class, property) IF p (class, property).
*properties (class, property) IF is-a (class, superclass),
                          properties (superclass, property).
```

All properties of the class *urbanResidence* can then be determined with the predicate

```
properties (UrbanResidence, prop).
```

which the following values for the variable prop fulfill:

```
prop = Resident
prop = Address
prop = Owner
```

In aggregation and association hierarchies, two types of property values occur: (1) values that are specifically owned by the composite object and, therefore, distinct and independent from those of its components and (2) values of the composite object which depend upon values of the properties of all components [Egenhofer 1986]. The mechanisms to describe such dependencies and ways to derive values is called *propagation* [Rumbaugh 1988]. Propagation guarantees consistency, because the dependent values of the aggregate are derived and need not be updated every time the components are changed. For example, the property *population* of the class *county* is the sum of the *populations* of all related instances of the class *settlement*.

LOBSTER was used for a prototype implementation of propagation. The following (simplified) facts describe the county Penobscot as an aggregate of two settlements Bangor and Orono-and some more in the rural areas-with the property `settlementPopulation.`

```
*p (Orono, SettlementPopulation, 10,000).
*p (Bangor, SettlementPopulation, 60,000).
*p (Orono, PartOf, Penobscot).
*p (Bangor, PartOf, Penobscot).
```

The population of the largest settlement in a county is derived from the settlements as the maximum of their populations. This dependency is expressed by the following rule, stating that the population of a specific county is the maximum of the population of all settlements which are part of it.

```
*propagates (Part0f, SettlementPopulation, PopulationofLargestSettlement, Maximum).
```

The generic rule for propagation is the following predicate. It describes the value of the property of an aggregate in terms of the values of the components using a specific aggregation function.

```
*p (aggregateClass, aggregateProperty, aggregateValue) IF propagates (relation,
 componentProperty, aggregateProperty, operation),
      p (componentClass, relation, aggregateClass),
      p (componentClass, componentProperty, componentValue),
      p (operation, componentValue, aggregateValue).
```

For example, the value of the property countyPopulation is then evaluated with

p (County, PopulationOfLargestSettlement, x).

and results in

```
x = 60,000
```

5.2 Geomorphology

The following experiment with LOBSTER describes how to define complex properties that can be derived from stored base properties. This example from geomorphology shows the definition of complex application-related terms in a rigorous manner so that users can understand them. These definitions are easy to program, but more important they make assumptions explicit so that different experts' opinions can be discussed. The distinction between different types of landscapes is evident for human observers, but at the same time they are difficult to express in formal terms. Verbal definitions of terms in natural language for geophysical phenomena have the substantial drawback that they are frequently based on other expressions which are not exactly defined, but are assumed to be generally understood [Frank 1986].

Symbolic processing for the extraction of geomorphologic features from landscape models has been proposed as a basis for formal analysis of terrain features [Palmer 1984]. This method uses a triangulated irregular network to describe a digital terrain model. In such a tessellation, nodes have an identifier and x, y, and z coordinates, and edges are described by an edge-identifier, the identifiers of the start and the end node, and the identifiers of the left and right area. The definition of terrain features is then based on the classification of an edge according to the downslopes of their adjacent triangles [Frank 1986]:

- an edge is *confluent* if the slope of both adjacent triangles is towards the edge;
- an edge is *diffluent* if the slope of both adjacent triangles is off the edge; and
- an edge is *transfluent* if the slope of one adjacent triangle is towards the edge and off the edge for the other triangle.

Two edges are connected if they share a common node and a *valley* is then a sequence of connected confluent edges.

These rules can be easily expressed as predicates in first-order predicate logic and implemented in a Prolog language [Robinson 1987b); however, pure Prolog [Clocksin 1981] lacks arithmetic operations, such as trigonometric functions, necessary to calculate the slope and determine the direction of flow over an edge. Such calculations can be easily performed in a traditional programming language, e.g., FORTRAN or Pascal, and then integrated with LOBSTER. The definitions given here can be directly executed. It is not necessary to manually translate them into code with the usual risk of introducing errors and misunderstandings. The predicates defined are also available in an interactive setting for experimentations.

```
*connectedEdge (el, e2) IF edge (ei, s, e) AND edge (e2, s, ee) AND
                    notEqual (ei, e2).
*connectedEdge (el, e2) IF edge (el, s, e) AND edge (e2, ss, e) AND
                    notEqual (el, e2).
*connectedEdge (el, e2) IF edge (el, s, e) AND edge (e2, e, ee) AND
                    notEqual (el, e2).
*connectedEdge (ei, e2) IF edge (el, s, e) AND edge (e2, ss, s) AND
                    notEqual (el, e2).
```

```
*confluentEdge (e)        IF edgeFlow  (e,    In, In).
*diffluentEdge (e)        IF edgeFlow  (e,    Out, Out).
*transfluentEdge (e)      IF edgeFlow  (e,    In, Out).
*transfluentEdge (e)      IF edgeFlow  (e,    Out, In).
```

The following two rules define a valley as a sequence of confluent edges and draw the resulting valley using the built-in predicate `drawEdge` as a co-routine.

```
*drawNextEdge (e) IF confluentEdge (e) AND drawEdge (e) AND
                          connectedEdge (e, ne) AND drawNextEdge (ne).
*drawValley (e) IF  confluentEdge (e) AND drawEdge (e) AND
                          connectedEdge (e, ne) AND drawNextEdge (ne).
```

5.3 Query Optimization

A common solution integrating multiple databases is the definition of a unifying query language which provides users with a view as if they dealt with a single system [Dayal 1986]. This is a likely scenario for all those GIS which use a special purpose data storage system for recording spatial data and a standard database management system for non-spatial data. Particularly important in such a distributed database environment is the determination of an efficient query processing strategy. The term *query optimization* refers to the process of calculating various strategies to process a specific query and selecting a plan which most likely provides the least expensive execution time. Various factors must be considered, such as the size of the database, the number of records involved in processing a particular operation, and the time to access records which may be distributed across different sites.

Query optimization is an important issue in a Prolog environment with large amounts of facts. Assume a rule of the form

```
a (x, G) IF b (x, z), c (z, G).
```

If the first predicate `b (x, z)` is a large database relation then it is not economical to use every fact stored to bind `x` and `z`, and then to try to prove the rest of the clause. A more sophisticated method must consider the approximate size of database relations and the existence of access paths [Warren 1981].

Based on the same principles as used in LOBSTER, a query optimizer has been implemented for a distributed spatial database [Hudson 1989]. It uses Horn clauses as an internal representation into which the user queries are translated and then applies rules to determine an optimal strategy. Since the sequence of predicates within a clause is immaterial to the logic of the clause, the predicates may be regrouped. This reordering is based on

- the (estimated) size of the relation for which a predicate stands,
- the estimated size of the result of a predicate,
- the estimated cost of verifying a predicate, and

the physical location of the data sets so that the transfer of data between various sites is minimized.

Query processing in logic databases [Bancilhon 1986] [Sagiv 1988] and rule-based query optimization [Freytag 1987] [Graefe 1987] are ongoing research topics.

6 Drawbacks of a Prolog-Based Query Language

Prolog was designed to express logical relations in a short-lived environment where users are aware of all facts and rules stored. Facts and rules are stored in files and users recall them explicitly when they need them. This approach is dramatically different from a database situation which is used over a long time and users do not remember all previously entered facts and rules. Furthermore, the database concept allows several users to share facts and rules in a multi-user environment.

6.1 Integrity Constraints

The schema definition in a database usually contains integrity constraints to prevent users from entering data which are not in accordance with the stated goals. This restriction is necessary so that users and application programs may rely on certain properties of the data. Violations of these rules produce incorrect results or fail to find data stored. Prolog contains no provisions to prevent the entry of invalid or contradictory data. Simple spelling errors in the name of a predicate while entering a rule will make that predicate

fail and the result will be "false." Such errors are extremely difficult to detect. If the database contains large numbers of facts, visual inspection by browsing is not possible anymore.

If an expert system should work for a long time, integrity constraints must be included and new data entered must be checked against them. Some examples may clarify this problem:

- Predicate names must be checked against previously used ones.
- New predicates are required to be explicitly declared by the user. Prolog implicitly declares a predicate with its first use, similar to the creation of variables with their first use in BASIC or FORTRAN and their known problems.
- In order to assist users in avoiding redundant declarations of the same or similar predicates [Kent 1981], have the user enter a description of the meaning of every new predicate declared. In addition, a query mechanism must be provided so that users may examine these descriptions later.
- The introduction of type constraints may help the checking of variables in predicates.

In LOBSTER, such assistance has been integrated and users have found them helpful.

6.2 Cyclic Rule Definitions

Systems using Prolog inference mechanisms are not well-protected against cyclic rule definitions, such as

```
a (x) IF b (x);
b (x) IF a W;
```

Collections of rules established for use during a short time, or used as a package and not expected to be expanded by the user, are generally checked by the programmer against cyclic rules.

LOBSTER contains some mechanisms that detect cycles; however, checking for cyclic structures during run time is costly in terms of execution time. More appropriate techniques for the detection of cyclic rules, for instance during input of new rules, are an issue of active research in the Prolog research community.

6.3 Order of Rules

Some Prolog programs rely on the order in which facts and rules are entered into the database. The order of facts should not disturb the execution of a Prolog program in a strictly logical sense. It may produce the results in a different order, but the results should be the same.

On the other hand, the order of rules is important for many recursive rules, especially if a specific stop rule (containing a *cut*) needs to be tested and the general recursive rule follows. For instance, the order of the two rules in the following example is crucial to correctly formulate `notEqual`-predicate. This rule says that if the two predicates x and y are equal, then `notEqual` is false; otherwise, `notEqual` is true.

```
notEqual (x, y) IF equal (x, y), cut, fail.
notEqual (x, y) IF .
```

If the order of the two rules was exchanged, the intended logic of the operation would have been changed.

```
notEqual (x, y) IF
notEqual (x, y) IF equal (x, y), cut, fail.
```

For any two predicates x and y the result would be true from the clause `notEqual (x, y) IF.`, and the second clause would never be tested.

It may be necessary to extend the data structure in figure 3 to include a class *program* consisting of several rules which are maintained and used in the given order.

7 Conclusion

We conclude that a GIS query language must provide a high-level abstraction of spatial data and geometric operations so that a user needs no explicit knowledge about their actual implementation; extensibility so that users may define new rules, maybe in the same system where data are stored and accessed; and recursion and loop constructs to formulate queries with transitive closure.

The use of Al methods and techniques for GIS are necessary to build flexible and powerful systems demanded by the user community. This paper reported on the integration of a specific Al method into a GIS programming environment. The result was LOBSTER, a persistent programming language based on Prolog. LOBSTER permitted us to study a number of areas in which Prolog and database techniques could be beneficial for GIS. The use of logic-based languages as GIS query languages has been explored as an alternative to the currently popular SQL type query languages.

LOBSTER was found to be powerful and flexible. Like any Prolog-based language, LOBSTER treats data and metadata in the same way; therefore, users may extend LOBSTER with appropriate rules whenever necessary. Furthermore, the extensibility of LOBSTER allows for definitions based on predicates that may be sometime difficult to implement in a pure first-order language. The implementation of additional built-in predicates which may be used within the language interface proved to be crucial for the implementation of propagation. Users' gained additional possibilities to access information in a GIS through this combination. An experimental system for geomorphologic feature detection demonstrated that LOBSTER can also be used to define specific interfaces for applications in an easy but comprehensive way.

8 Acknowledgements

Over the years, many colleagues and friends of ours helped with and contributed to the implementation of LOBSTER and we want to thank them all. Particularly appreciated were many discussions with Vince Robinson on the use of Al for GIS. Doug Hudson and Bruce Palmer worked on the applications we used as examples in this paper. The team implementing LOBSTER included R. Michael White and Eric Carlson.

References

[Abler 1987] R. Abler. The National Science Foundation National Center for Geographic Information and Analysis. International Journal of Geographical Information Systems, 1(4):303-326, 1987.

[ANSI 1986] X3112-86-2 American National Standard Database Language SQL. American National Standards Institute, January 1986.

[Aronson] 9831 P. Aronson and S. Morehouse. The ARC/INFO Map Library: A Design for a Digital Geographic Database. In: Auto-Carto VI, pages 346-382, Ottawa, 1983.

[Bancilhon 1986] F. Bancilhon and others. Magic Sets and Other Strnage Ways to Implement Logic Programs. In: ACM Symposium on Principles of Database Systems, pages 1-15, Cambridge, MA, March 1986.

[Barr 1982] A. Barr and E. Feigenbaum. The Handbook of Artificial Intelligence. Pitman, London,1982.

[Borgida 1984] A. Borgida and others. Generalization/Specialization as a Basis for Software Specification. In: M. Brodie, J. Mylopoulos, and J. Schmidt, editors, On Conceptual Modelling, pages 87-114, Springer Verlag, New York, NY, 1984.

[Brodie 1981] M. Brodie. Association: A Database Abstraction for Semantic Modeling. In: 2nd International Entity-Relationship Conference, Washington, D.C., 1981.

[Brodie 1984] M. Brodie, J. Mylopoulos, and J. Schmidt, editors. On Conceptual Modelling, Perspectives from Artificial Intelligence, Databases, and Programming Languages. Springer Verlag, New York, NY, 1984.

[Cardelli 19841 L. Cardelli. A Semantics of Multiple Inheritance. In: G. Kahn, D. McQueen, and G. Plotkin, editors, Semantics of Data Types, pages 51-67, Springer Verlag, New York, NY, 1984.

[Chamberlin 1976] D. Chamberlin, M. Astrahan, K. Eswaran, P. Griffiths, R. Lorie, J. Mehl, P. Reisner, and B. Wade. SEQUEL 2: A Unified Approach to Data Definition, Manipulation, and Control. IBM Journal of Research and Development, 20(6):560-575, November 1976.

[Clocksin 1981] W.F. Clocksin and C.S. Mellish. Programming in Prolog. Springer Verlag, New York, NY, 1981.

[Codd 1970] E.F. Codd. A Relational Model for Large Shared Data Banks. Conununications of the ACM, 13(6):377-387, June 1970.

[Codd 1972] E.F. Codd. Relational Completeness of Data Base Sublanguages. In: R. Rustin, editor, Data Base Systems, pages 65-98, Prentice Hall, Englewood Cliffs, NJ, 1972.

[Dahl 1966] O.-J. Dahl and K. Nygaard. SIMULA-An Algol-based Simulation Language. Communications of the ACM, 9(9):671-678, September 1966.

[Dayal 1986] U. Dayal. Query Processing in a Multidatabase System. In: W. Kim, D. Reiner, and D. Batory, editors, Query Processing in Database Systems, pages 81-108, Springer Verlag, New York, NY, 1986.

[Dittrich 1986] K. Dittrich. Object-Oriented Database Systems: The Notation and The Issues. In: K. Dittrich and U. Dayal, editors, International Workshop in ObjectOriented Database Systems, Pacific Grove, CA, pages 2-4, IEEE Computer Society Press, Washington, D.C., 1986.

[Egenhofer 1986] M. Egenhofer and A. Frank. Connection between Local and Regional: Additional 'Intelligence' Needed. In: FIG XVIII. International Congress of Surveyors, Commission 3, Land Information Systems, Toronto, Ontario, Canada, 1986.

[Egenhofer 1988] M. Egenhofer and A. Frank. Towards a Spatial Query Language: User Interface Considerations. In: D. DeWitt and F. Bancilhon, editors, 14th International Conference on Very Large Data Bases, pages 124-133, Los Angeles, CA, August 1988.

[Egenhofer 1989a] M. Egenhofer and A. Frank. Object-Oriented Modeling in GIS: Inheritance and Propagation. In: AUTO-CARTO 9, Ninth International Symposium on Comp u ter- Assisted Cartography, pages 588-598, Baltimore, MD, April 1989.

[Egenhofer 1989b] M. Egenhofer and A. Frank. PANDA: An Extensible DBMS Supporting Object-Oriented Software Techniques. In: T. Harder, editor, Database Systems in Office, Engineering, and Science, pages 74-79, Springer-Verlag, New York, NY, March 1989.

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

[Frank 1981] A. Frank. Applications of DBMS to Land Information Systems. In: C. Zaniolo and C. Delobel, editors, Seventh International Conference on Very Large Data Bases, pages 448-453, Cannes, France, 1981.

[Frank 1982a] A. Frank. MAPQUERY-Database Query Language for Retrieval of Geometric Data and its Graphical Representation. ACM Computer Graphics, 16(3):199-207, July 1982.

[Frank 1982b] A. Frank. PANDA-A Pascal Network Database System. In: G.W. Gorsline, editor, Fifth Symposium on Small Systems, Colorado Springs, CO, 1982.

[Frank 1984] A. Frank. Extending a Database with Prolog. In: L. Kerschberg, editor, First International Workshop on Expert Database Systems, pages 665-676, Kiawah Island, SC, October 1984.

[Frank 1986] A. Frank, B. Palmer, and V. Robinson. Formal Methods for the Accurate Defintion of some Fundamental Terms in Physical Geography. In: D. Marble, editor, Second International Symposium on Spatial Data Handling, pages 583599, Seattle, WA, 1986.

[Frank 1988] A. Frank. Requirements for a Database Management System for a GIS. Photogrammetric Engineering & Remote Sensing, 54(11):1557-1564, November 1988.

[Freytag 1987] J.C. Freytag. A Rule-Based View of Query Optimization. In: SIGMOD Conference, pages 172-180, San Francisco, CA, May 1987.

[Gallaire 1984a] H. Gallaire and J. Minker, editors. Logic and Data Bases. Plenum Press, New York, NY, 1984.

[Gallaire 1984b] H. Gallaire, J. Minker, and J. Nicolas. Logic and Databases: A Deductive Approach. ACM Computing Surveys, 16(2):153-185, June 1984.

[Goldberg 1983] A. Goldberg and D. Robson. Smalltalk-80. Addison-Wesley Publishing Company, Reading, MA, 1983.

[Graefe 1987] G. Graefe and D. DeWitt. The EXODUS Optimizer Generator. In: SIG-MOD Conference, pages 160-171, San Francisco, CA, May 1987.

[Gilting 1988] R. Giiting. Geo-Relational Algebra: A Model and Query Language for Geometric Database Systems. In: J. Schmidt, S. Ceri, and M. Missikoff, editors, Advances in Database Technology-EDBT '88, International Conference on Extending Database Technology, Venice, Italy, pages 506-527, Springer Verlag, New York, NY, 1988.

[Harder 1985] T. Harder and A. Reuter. Architecture of Database Systems for Non-Standard Applications (in German). In: A. Blaser and P. Pistor, editors, Database Systems in Office, Engineering, and Science, pages 253-286, Springer Verlag, New York, NY, March 1985.

[Hayes-Roth 1983] F. Hayes-Roth and others. Building Expert Systems. Addison-Wesley Publishing Company, Reading, MA, 1983.

[Hudson 1989] D. Hudson. A Unifying Database Formalism. In: ASPRS/ACSM Annual Convention, pages 146-153, Baltimore, MD, April 1989.

[Jarke 1984] M. Jarke, J. Clifford, and Y. Vassiliou. An Optimizing Front-End to a Relational Query System. In: B. Yormark, editor, Annual Meeting ACM SIGMOD, pages 296-306, Boston, MA, June 1984.

[Kent 1981] W. Kent. Data Model Theory Meets a Practical Application. In: C. Zaniolo and C. Delobel, editors, Seventh International Conference on Very Large Data Bases, Cannes, France, 1981.

[Kowalski 1979] R. Kowalski. Logic for Problem Solving. Elsevier Science Publishing Co., New York, NY, 1979.

[Laurini 1989] R. Laurini and F. Milleret. Solving Spatial Queries by Relational Algebra. In: AUTO-CARTO 9, Ninth International Symposium on Computer-Assisted Cartography, pages 426-435, Baltimore, MD, April 1989.

[Luk 1986] W.S. Luk and S. Kloster. ELFS: English Language From SQL. ACM Transactions on Database Systems, 11(4):447-472, December 1986.

[McKeown 1987] D. McKeown. The Role of Artificial Intelligence in the Integration of Remotely Sensed Data with Geographic Information Systems. IEEE Transactions on Geoscience and Remote Sensing, 25(3):330-348, May 1987.

[Motro 1984) A. Motro. Browsing in a Loosely Structured Database. In: B. Yormark, editor, Annual Meeting ACM SIGMOD, pages 197-207, Boston, MA, June 1984.

[Mylopoulos 1981] J. Mylopoulos. An Overview of Knowledge Representation. SIGMOD Record, 11(2):5-12, 1981.

[Mylopoulos 1984] J. Mylopoulos and H. Levesque. An Overview of Knowledge Representation. In: M.L. Brodie and others, editors, On Conceptual Modelling, pages 3-17, Springer Verlag, New York, NY, 1984.

[Palmer 1984] B. Palmer. Symbolic Feature Analysis and Expert Systems. In: International Symposium on Spatial Data Handling, pages 465-478, Zurich, Switzerland, August 1984.

[Peuquet 1987] D. Peuquet. Research Issues in Artificial Intelligence and Geographic Information Systems. In: International Geographic Information Systems (IGIS) Symposium: The Research Agenda, pages 119-127, Arlington, VA, November 1987.

[Reisner 1981] P. Reisner. Human Factors Studies of Database Query Languages: A Survey and Assessment. ACM Computing Surveys, 13(l):13-31, March 1981.

[Robinson 1987a] V. Robinson and A. Frank. Expert Systems for Geographic Information Systems. Photogrammetric Engineering & Remote Sensing, 53(10):1435-1441, October 1987.

[Robinson 1987b] V. Robinson and others. Expert Systems for Geographic Information Systems in Resource Management. Al Applications in Natural Resource Management, 1(1);47-57, 1987.

[Rumbaugh 1988] J. Rumbaugh. Controlling Propagation of Operations using Attributes on Relations. In: OOPSLA '88, pages 285-296, San Diego, CA, September 1988.

[Sagiv 1988] Y. Sagiv. Optimizing DATALOG Programs. In: J. Minker, editor, Foundations of Deductive Databases and Logic Programming, pages 659-698, Morgan Kaufmann, Los Altos, CA, 1988.

[Smith 1977] J.M. Smith and D.C.P. Smith. Database Abstractions: Aggregation. Communications of the ACM, 20(6):405-413, June 1977.

[Smith 1987] T. Smith, D. Peuquet, S. Menon, and P. Agrawal. KBGIS-11: A KnowledgeBased Geographical Information System. International Journal of Geographical Information Systems, 1(2):149-172, 1987.

[Stonebraker 1976] M. Stonebraker, E. Wong, and P. Kreps. The Design and Implementation of INGRES. ACM Transactions on Database Systems, 1(3):189-222, September 1976.

[Stroustrup 1986] B. Stroustrup. The C++ Programming Language. Addison-Wesley Publishing Company, 1986.

[Vassilou 1983] Y. Vassilou, J. Clifford, and M. Jarke. How Does an Expert System get its Data? In: M. Schkolnick and C. Thanos, editors, Nineth International Conference on Very Large Data Bases, pages 70-72, Florence, Italy, 1983.

[Warren 1981] D.H.D. Warren. Efficient Processing of Interactive Relational Database Queries Expressed in Logic. In: C. Zaniolo and C. Delobel, editors, Seventh International Conference on Very Large Data Bases, pages 272-281, Cannes, France, 1981.

[Zloof 1977] M.M. Zloof. Query-by-Example: A Database Language. IBM Systems Journal, 16(4):324-343, 1977.