Spatial SQL: A Query and Presentation Language

Max J. Egenhofer, *Member, IEEE*

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. It consists of two components, a *query language* to describe what information to retrieve and a *presentation language* to specify how to display query results. 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 the Graphical Presentation Language GPL to manipulate or examine the graphical presentation.

Index Terms—Geographic Information Systems, graphical presentation, query languages, query result combinations, spatial context, spatial databases, SQL, topological relationships.

1 Introduction

Recently, attention has been focused on spatial databases [34, 56] which combine conventional and spatially related data. The need for spatial query languages has been identified in several different application domains of spatial databases such as Geographic Information Systems (GIS's) [30], image databases [10], or remote sensing [45]. 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 [9], Quel [60], or Query-by-Example [63] 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 seeing the treatment of reals as bit strings. Examples are GEO-QUEL [5, 33], a QUEL extension, Query-by-Pictorial-Example [11], a Query-by-Example extension, or a Quel extension for image processing [25]. Spatial data have additional properties, such as *geometry* and *graphical presentation*, 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 [28, 49, 53] and the relational algebra has been extended with spatial operations to a Geo-Relational Algebra [35]. 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 presentation with which interactive communication is encouraged [36], or GEOBASE [4] 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 will be reviewed at more detail in section II.C.

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 [21]. 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 *presentation* of spatial data is also considered to be part of a

spatial query language.

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 [57], TOSQL [2], and HSQL [54] as respective extensions of QUEL and SQL, and SQL dialects for date and time [14] and dynamic complex objects [47]. The particular combination of the description of retrieval and graphical presentation 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 [1]. It is completed by the *Graphical Presentation Language (GPL)* [20], 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 embedding of Spatial SQL into a human interface [21], query processing and optimization [39], and the state of the implementation [17] have been described elsewhere.

The remainder of this paper is organized as follows: Section II evaluates previous SQL-based 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 presentation of query results. The architecture of an SQL-based spatial query and presentation language is discussed in section III. Sections IV and V present the syntax and semantics of the retrieval language Spatial SQL and the Graphical Presentation Language GPL, respectively. An example in section VI shows the use of Spatial SQL and GPL. Finally, in section VII our conclusions are presented.

2 Spatial Query Languages

Spatial SQL is based upon the relational database query language SQL [9]. An SQL query uses the SELECT–FROM–WHERE clause for the three operations of relational algebra projection, Cartesian product, and selection, respectively. In ad-

dition, aggregate functions, such as sum, minimum, or average, may be employed to calculate a single value from a set of tuples [7].

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 [1]. 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 non-spatial data which will be the subject of user queries. Three fundamental categories of queries in a spatial information system can be distinguished [4]:

- Queries exclusively about spatial properties, e.g., "Retrieve 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., "Retrieve all neighbors of the parcel located at 26 Grove Street."

It is crucial for spatial query languages to provide syntactical means for all three categories. Traditional query languages do so 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 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 critique of SQL [12] 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 presentation 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 [48]. Additional clauses for the treatment of new concepts are undesirable. Although such extensions—adding a clause for graphical presentation, 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 eleven crucial requirements for a spatial query language which are not covered by conventional systems [21]. 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 x-y coordinates [52].

2. The display of query results in graphical form, as the most natural form to analyze spatial data, allows for the presentation 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. The possible combination of one query result with the results of one or more previous queries gives rise to a dynamic interaction.

4. Graphical presentations 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.

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

6. 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 [32].

7. Varying graphical presentation of spatial objects and their parts described in terms of colors, patterns, intensity, and symbols [6] 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 graphical presentation.

9. Labels play an important role in the understanding of drawings and users must be able to select objects to be labeled from the query language.

10. Users of conventional maps have developed significant skills to interpret the actual size of objects drawn and the selection of a specific scale of a rendering allows users of computerized spatial information systems to produce maps on which they can continue to apply their capabilities.

11. Frequently, users work on a subscene, not the whole spatial database, and the query language must offer them tools to restrict the area of interest to a particular geography to which all upcoming queries will refer.

Several proposals have been made to make SQL applicable as a query language for spatial databases by incorporating solutions for some of these requirements. The most significant extensions will be reviewed subsequently.

2.3 SQL-Based Query Languages for Spatial Databases

GEOQL [51] fully preserves the SQL structure and adds to the standard definitions of SQL the concept of geometry in terms of the bounding lines of spatial objects, spatial operators between geographic objects, and windows.

Another proposal [55] includes extensions for the treatment of spatial relations and a picture list to manage 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 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 [53]. Each spatial object is extended by an attribute *loc* which 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 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 an update of this language the logical consequence was drawn and the AT clause was merged with the WHERE clause [52]. Predefined output formats are very useful, but the user also needs a language to manipulate the graphical presentation.

The SQL-based query language for the Geographic Information System KGIS [43] is a combination of syntax extensions in the SELECT–FROM–WHERE clauses

and a command set outside of SQL [40]. 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 SELECT–FROM–WHERE query.

The query language for the TIGRIS GIS [37] pursues an object-oriented approach, giving up some SQL principles [38]. 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 1 where the languages are analyzed against the eleven requirements for a spatial query language.

3 Separating Retrieval and Display Instructions

The production of a map with a single instruction may be of interest [24], 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 presentation style and (2) edit the graphical rendering frequently by modifying only the graphical "parameters" of objects already displayed [19]. 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 [27] or labels may be placed outside of the visible screen area [29, 31].

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 *presentation language* releases the user from the need of describing the graphical presentation with each query. Instead, a *display environment* is maintained that manages the state of the graphical presentation and applies it to the query result. Query and display specifications then interact closely: the presentation 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 presentation 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 Presentation 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 Presentation Language GPL, a dedicated language for the description of the graphical presentation which was designed as a superset of the spatial query language. Fig. 1 shows the three standard situations of a user interacting with GPL and Spatial SQL: a user modifies the display environment with one or several GPL

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 GPL 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.

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 spatial operations and relationships, and a method to refer to objects on drawings by pointing at them with a direct manipulation device are introduced. To date, only subsets of a spatial algebra [35, 61] and formal definitions of spatial relationships [18] 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 [46] is cumbersome and syntactically very complex [62]. 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 the Euclidean 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 criteria.

Spatial SQL provides a higher-level abstraction of spatial data and extends the domains in the relational calculus by four spatial domains—*spatial_0*, *spatial_1*, *spatial_2*, and *spatial_3*—for 0-, 1-, 2-, and 3-dimensional spatial objects, respectively. These domains are generalized to a dimension-independent domain *spatial*. An attribute over such 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 [23, 42]; however, they are considered as selection criteria, similar to traditional predicates, and can be used similarly to conventional relationships. Only the specified spatial operations and relationships apply to spatial attributes.

4.2 Spatial Operations and Relationships

Unary spatial operations access a spatial property of a tuple of a spatial relation. 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 O, i.e., all object parts in the boundary of O [16]. Complementary to boundary, *interior* calculates the interior faces, i.e., all those object parts that are not in the boundary of O. 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 1- and 2-dimensional objects, the set consists of the operations *boundary*, *boundingNodes*, *boundingEdges*, *interior*, *interiorNodes*, *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* [13], *complement* [8], and *convex hull* [35], fit into this concept and may be easily incorporated into Spatial SQL.

Binary spatial operations calculate a value among two tuples of spatial relations, similar 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 onto 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.

Binary spatial relationship are relationships between two spatial attributes. They 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 [18, 22]. For example, the neighborhood relationship between two areas can be expressed in terms of the four boundary and interior intersections (Table 2). 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*.

4.3 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 graphical presentation of an object at different scales and levels of detail. For example, a town on a map may be presented 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.

4.4 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 [19]. Ambiguities are reduced since the user points to a location and specifies in the query which kind of object, i.e., relation name, to select. On a map with a partition of counties, displayed by their boundaries, and their major cities displayed as small circles, a user asks for the the name of a particular city by clicking to its map symbol and entering the following query:

```
SELECT name
FROM   city
WHERE  geometry = PICK;
```

On the other hand, if one points to the same location and specifies that the object selected is a county, the county name will be retrieved:

```
SELECT name
FROM   county
WHERE  geometry = PICK;
```

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 [21].

5 The Graphical Presentation Language GPL

The Graphical Presentation Language GPL [20] provides tools for the manipulation of the graphical presentation of query results. Central to GPL is the concept of the *display environment* which handles the information about how to display query results. During query processing, this information is integrated with the user query so that the query result is rendered according the display description. Unless the user changes the environment with a GPL instruction, the display environment continues to produce with each query a map of the same style.

GPL offers instructions to SET a graphic specification, establishing the characteristics of the display environment; to CANCEL a specification, resetting the display environment; and to examine (SHOW) the current values of the display environment. Environments established with GPL are effective for all subsequent Spatial SQL queries and stay valid until they are cancelled or overwritten with a corresponding instruction. SET and CANCEL can be qualified with PERMANENT so that the instructions persist across sessions, and with IMMEDIATELY to allow users to update the graphical presentation of the current map prior to asking the next query.

Six types of graphic specification are distinguished: (1) the display mode, (2) the graphical presentation, (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. A major impediment for any SQL-based query language with more than one kind of rendering is that the SELECT clause is overloaded with the projection of the attributes onto the resulting relation and the implied tabular representation [15]. Spatial SQL solves this problem by the separation of the two issues so that the SELECT command performs only the relational projection operation, and the query result will be displayed according to the current display mode selected with GPL. The six display modes are the conventional alphanumeric display and five graphical display types [19]: (1) NEW, starting a new drawing; (2) OVERLAY, adding the result onto an existing drawing; (3) REMOVE, erasing the result from a drawing; (4) INTERSECT, determining the common ob-

jects on the display and in the query result; and (5) 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 [31].

5.2 Visual Variables

The graphical presentation plays a more important role for spatial data than for the presentation of lexical data in tabular form [19]; therefore, the use of *visual variables* (colors pattern, and symbols) [6] to specify the graphical presentation of spatial objects is included into GPL. These graphic attributes can be specified for either an entire spatial relation or instances of spatial relations which fulfill specific constraints. This lets different users view a spatial database from their individual perspectives and allows them to tailor the graphical presentation of query results to their specific needs. Complementary to defining these visual variables, users can check the current settings by "looking at the legend," i.e., inquiring about the graphical presentation.

This part of GPL depends upon the user's hardware; therefore, the set of terms for the visual variables is is designed to be extensible so that it can be adapted to different environments.

5.3 Scale and Window

The scale of the graphical presentation 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 $1 : n$. 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 a graphical presentation is extremely dependent on the context and environment in which it is shown. Unlike lexical presentation, it is often in-

sufficient to draw only those objects directly requested in the query. The graphical presentation 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 [21]. GPL 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. 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";
```

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 Conclusions

Spatial SQL is characterized by minimal extensions to SQL and the introduction of the Graphical Presentation Language (GPL). The eleven 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 presentation directed from GPL; (3) the display modes NEW, OVERLAY, REMOVE, INTERSECT, and HIGHLIGHT in GPL to combine query results into a single drawing; (4) the definition of CONTEXT in GPL; (5) the examination of CONTENT from GPL; (6) selection by pointing via the PICK qualifier in WHERE clauses; (7) the manipulation of the graphical presentations of objects with colors, patterns, and

symbols; (8) the examination of the map LEGEND in GPL; (9) the combination of both graphical and alphanumeric data in a single result which enable objects to be labeled; (10) graphical presentations 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 [26], HDBL [44], and the O_2 query language [3]. 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 [14, 50, 59] 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 presentation of query results.

Spatial SQL and GPL 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 [21]. 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 [41]. Likewise, the implementation of GPL 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 [58]. 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.

Acknowledgments

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

[1] "X3.135-1986 American National Standard Database Language SQL," American National Standards Institute, Jan. 1986.

[2] G. Ariav, "A temporally oriented data model," *ACM Trans. Database Syst.*, vol. 11, no. 4, pp. 499–527, Dec. 1986.

[3] F. Bancilhon, S. Cluet, and C. Delobel, "A query language for the O_2 object-oriented database system," in *Proc. 2nd Int. Workshop Database Programming Languages*, June 1989, pp. 122–138.

[4] R. Barrera and A. Buchmann, "Schema definition and query language for a geographical database system," *IEEE Trans. Comp. Architecture: Pattern Analysis and Image Database Management*, vol. 11, pp. 250–256, 1981.

[5] R. Berman and M. Stonebraker, "Geo-Quel, a system for the manipulation and display of geographic data," *ACM Comp. Graphics*, vol. 11, no. 2, pp. 186–191, 1977.

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

[7] G. von Bültzingsloewen, "Translating and optimizing SQL queries having aggregates," in *Proc. 13th Int. Conf. VLDB*, Brighton, England, Sep. 1987, pp. 235–243.

[8] W. Burton, "Logical and physical data types in geographic information systems," *Geo-Processing*, vol. 1, no. 2, pp. 167–181, 1979.

[9] 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 J. Res. and Develop.*, vol. 20, no. 6, pp. 560–575, Nov. 1976.

[10] S.K. Chang and T. Kunii, "Pictorial database systems," *Computer*, vol. 14, no. 11, pp. 13–21, Nov. 1981.

[11] N.S. Chang and K.S. Fu, "Query-by-pictorial-example," *IEEE Trans. Software Eng.*, vol. SE-6, no. 6, pp. 519–524, Nov. 1980.

[12] E.F. Codd, "Fatal flaws in SQL," *Datamation*, vol. 34, no. 16, Aug. 1988.

[13] N. Cox, B. Alfred, and D. Rhind, "A relational data base system and a proposal for a geographic data type," *Geo-Processing*, vol. 1, no. 3, pp. 217–229, 1979.

[14] C.J. Date, "Defining data types in a database language," *SIGMOD Rec.*, vol. 17, no. 2, pp. 53–76, June 1988.

[15] M. Egenhofer, "An extended SQL syntax to treat spatial objects," in *Proc. 2nd Int. Seminar Trends and Concerns of Spatial Sciences*, Fredericton, New Brunswick, June 1987, pp. 83–95.

[16] M. Egenhofer, "Graphical representation of spatial objects: an object-oriented view," Tech. Rep. 83, Department of Surveying Engineering, University of Maine, Orono, ME, July 1988.

[17] M. Egenhofer, "Spatial query languages," PhD thesis, University of Maine, Orono, ME, 1989.

[18] M. Egenhofer, "A formal definition of binary topological relationships," in *Proc. 3rd Int. Conf. Foundations of Data Organization and Algorithms*, Paris, France, June 1989, LNCS vol. 367, New York, NY: Springer-Verlag, pp. 457–472.

[19] M. Egenhofer, "Interaction with geographic information systems via spatial queries," *J. Visual Languages and Computing*, vol. 1, no. 4, pp. 389–413, 1990.

[20] M. Egenhofer, "Extending SQL for cartographic display," *Cartography and Geographic Information Systems*, 1991 (in press).

[21] M. Egenhofer and A. Frank, "Towards a spatial query language: user interface considerations," in *Proc. 14th Int. Conf. VLDB*, Long Beach, CA, Aug. 1988, pp. 124–133.

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

[23] M. Egenhofer and J. Herring, "A mathematical framework for the definition of topological relationships," in *Proc. Fourth Int. Symp. Spatial Data Handling*, Zurich, Switzerland, July 1990, pp. 814–819.

[24] H.-D. Ehrich, F. Lohmann, K. Neumann, and I. Ramm, "A database language for scientific map data," *Geologisches Jahrbuch*, vol. A, no. 104, pp. 139–152, 1988.

[25] D.W. Embley and G. Nagy, "Toward a high-level integrated image database system," Tech. Rep., Image Processing Laboratory, Rensselaer Polytechnic Institute, Troy, NY, Apr. 1986.

[26] D. 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 Trans. Office Inform. Syst.*, vol. 5, no. 1, pp. 48–69, Jan. 1986.

[27] A. Frank, "Applications of DBMS to land information systems," in *Proc. 7th Int. Conf. VLDB*, Cannes, France, Aug. 1981, pp. 448–453.

[28] A. Frank, "Mapquery—database query language for retrieval of geometric data and its graphical representation," *ACM Comp. Graphics*, vol. 16, no. 3, pp. 199–207, July 1982.

[29] A. Frank, "Computer assisted cartography—graphics or geometry," *J. Surveying Engineering*, vol. 110, no. 2, pp. 159–168, Aug. 1984.

[30] A. Frank, "Requirements for a database management system for a GIS," *Photogrammetric Eng. & Remote Sensing*, vol. 54, no. 11, pp. 1557-1564, Nov. 1988.

[31] H. Freeman and J. Ahn, "On the problem of placing names in a geographic map," *Int. J. Pattern Recognition and Art. Intell.*, vol. 1, no. 1, pp. 121–140, 1987.

[32] M. Friedell, J. Barnett, and D. Kramlich, "Context-sensitive, graphic presentation of information," *ACM Comp. Graphics*, vol. 16, no. 3, pp. 181–188, July 1982.

[33] A. Go, M. Stonebraker, and C. Williams, "An approach to implementing a geo-data system," Tech. Rep., Memo ERL-M529, Electronics Research Laboratory, University of California, Berkeley, CA, June 1975.

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

[35] R. Güting, "Geo-relational algebra: a model and query language for geometric database systems," in *Proc. Int. Conf. Extending Database Technology*, Venice, Italy, May 1988, LNCS vol. 303, New York, NY: Springer Verlag, pp. 506–527.

[36] C. Herot, "Spatial management of data," *ACM Trans. Database Syst.*, vol. 5, no. 4, pp. 493–513, Dec. 1980.

[37] J. Herring, "TIGRIS: topologically integrated geographic information systems," in *Proc. AUTO-CARTO 8*, Baltimore, MD, March 1987, pp. 282–291.

[38] J. Herring, R. Larsen, and J. Shivakumar, "Extensions to the SQL language to support spatial analysis in a topological data base," in *Proc. GIS/LIS '88*, San Antonio, TX, Nov. 1988, pp. 741–750.

[39] D. Hudson, "Combined Spatial/II and RDB database operation: query coordination and optimization strategies," Tech. Rep., University of Maine, Orono, Department of Surveying Engineering, Orono, ME, 1988.

[40] K. Ingram and W. Phillips, "Geographic information processing using a SQL-based query language," in *Proc. AUTO-CARTO 8*, Baltimore, MD, March 1987, pp. 326–335.

[41] J. Jackson, "Developing an effective human interface for geographic information systems using metaphors," in *Proc. ACSM-ASPRS Annual Conv.*, Denver, CO, March 1990, pp. 117–125.

[42] W. Kainz, "Spatial relationships—topology versus order," in *Proc. Fourth Int. Symp. Spatial Data Handling*, Zurich, Switzerland, July 1990, pp. 814–819.

[43] T. Keating, W. Phillips, and K. Ingram, "An integrated topologic database design for geographic information systems," *Photogrammetric Eng. & Remote Sensing*, vol. 53, no. 10, pp. 1399–1402, Oct. 1987.

[44] V. Linnemann, K. Küspert, P. Dadam, P. Pistor, R. Erbe, A. Kemper, N. Südkamp, G. Walch, and M. Wallrath, "Design and implementation of an extensible database management system supporting user defined data types and functions," in *Proc. 14th Int. Conf. VLDB*, Long Beach, CA, Aug. 1988, pp. 294–305.

[45] G. Lohman, J. Stoltzfus, A. Benson, M. Martin, and A. Cardenas, "Remotely-sensed geophysical databases: experience and implications for generalized DBMS," in *Proc. SIGMOD 83*, San Jose, CA, 1983, pp. 146–160.

[46] R. Lorie, and A. Meier, "Using relational dbms for geographical databases," *Geo-Processing*, vol. 2, pp. 243–257, 1984.

[47] R. Lorie and H.-J. Schek, "On dynamically defined complex objects and SQL," in *Proc. 2nd Int. Workshop Object-Oriented Database Syst.*, Bad Münster am Stein-Ebernburg, Germany, Sep. 1988, LNCS vol. 334, New York, NY: Springer-Verlag, pp. 323–328.

[48] W.S. Luk and S. Kloster, "ELFS: english language from SQL," *ACM Trans. Database Syst.*, vol. 11, no. 4, pp. 447–472, Dec. 1986.

[49] D. McKeown, "MAPS: the organization of a spatial database system using imagery, terrain, and map data," Tech. Rep. CMU-CS-83-136, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, July 1983.

[50] J. Ong, D. Fogg, and M. Stonebraker, "Implementation of data abstraction in the relational database system INGRES," *SIGMOD Rec.*, vol. 14, no. 1, March 1984.

[51] B.C. Ooi, R. Sacks-Davis, and K. McDonell, "Extending a DBMS for geographic applications," in *Proc. IEEE 5th Int. Conf. Data Eng.*, Los Angeles, CA, Feb. 1989, pp. 590–597.

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

[53] N. Roussopoulos and D. Leifker, "Direct spatial search on pictorial databases using packed R-trees," in *Proc. SIGMOD 85*, Austin, TX, May 1985, SIGMOD Rec., vol. 14, no. 4, pp. 17–31.

[54] N. Sarda, "Extensions to SQL for historical databases," *IEEE Trans. Knowledge and Data Eng.*, vol. 2, no. 2, pp. 220–230, June 1990.

[55] A. Sikeler, "Examination of storage structures for 3-dimensional objects (in German)," Tech. Rep., University Kaiserslautern, 1985.

[56] T. Smith and A. Frank, "Very large spatial databases: report from the specialist meeting," *J. Visual Languages and Computing*, vol. 1, no. 3, pp. 291–309, 1990.

[57] R. Snodgrass, "The temporal query language TQUEL," *ACM Trans. Database Syst.*, vol. 12, no. 6, pp. 247–298, June 1987.

[58] D. Steiner, M. Egenhofer, and A. Frank, "An object-oriented carto-graphic output package," in *Proc. ASPRS-ACSM Annual Conv.*, Baltimore, MD, March 1989, pp. 104–113.

[59] M. Stonebraker, B. Rubenstein, and A. Guttman, "Application of abstract data types and abstract indices to CAD databases," in *Proc. ACM SIGMOD Conf. Eng. Design Applications*, San Jose, CA, 1983, pp. 107–113.

[60] M. Stonebraker, E. Wong, and P. Kreps, "The design and implementation of INGRES," *ACM Trans. Database Syst.*, vol. 1, no. 3, pp. 189–222, September 1976.

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

[62] A. Westlake and I. Kleinschmidt, "The implementation of area and membership retrievals in point geography using SQL," in *Proc. 5th Int. Conf. Statistical and Scientific Database Management*, Charlotte, NC, Apr. 1990, LNCS vol. 420, New York, NY: Springer-Verlag, pp. 200–218.

[63] M.M. Zloof, "Query-by-example: a database language," *IBM Syst. J.*, vol. 16, no. 4, pp. 324–343, 1977.