Restructuring the SQL Framework for Spatial Queries

Bo Huang and Hui Lin

Department of Geography & Joint Laboratory for Geoinformation Science
The Chinese University of Hong Kong
Shatin, NT, Hong Kong

Abstract

This paper presents an approach to designing a spatial query language, called GeoSQL, in terms of the conventional spatial query and implementation process. A critical factor to the design is how to accommodate spatial operators in an appropriate form, while being compatible with the Structured Query Language (SQL) standard. To achieve this, the FROM clause of SQL is restructured to contain spatial operators via a subquery so that the results of spatial operations can be easily fed into both the SELECT and WHERE clauses. The subquery in the FROM clause creates an intermediate relation, on which the selection in terms of certain criteria is conducted. This is a distinct characteristic of GeoSQL. The syntax and semantics of GeoSQL are described, and a set of examples for testing the expressiveness of the language is given. The interface of the language is also designed with the introduction of visual constructs (e.g., icons and ListBoxes) to aid the entry of query text. This distinguishes GeoSQL's interface from the previous extended SQLs, which only employ pure text for constructing a query. After this, an implementation of GeoSQL is discussed. This paper finally suggests further extending GeoSQL for temporal and fuzzy queries.

摘 要

本文根据常规的空间分析过程，设计了空间查询语言：GeoSQL。GeoSQL采用了子查询(subquery)，重组了FROM语句，这是GeoSQL的主要特色之一。其次，GeoSQL的界面设计融入了Icon，ListBox等可视化部件，使查询文本输入变得容易，同时减少了语法错误，这是GeoSQL的另一特色。本文介绍了GeoSQL的表达形式、界面设计和实现方法。

I. INTRODUCTION

The need for a formal spatial query language has been widely identified in GIS community [8, 10]. Therefore, several approaches to devising a spatial query language have been developed [2, 4]. Of them, extending the relational database languages, primarily SQL, is a major one.

SQL, very suitable for the retrieval of lexical data, has been the standard query language for relational databases [1]. However, based on the underlying power of relational algebra, SQL has proved to be insufficient for the queries involving spatial proprieties such as metric and topology [7]. Hence a variety of extended SQLs were addressed (e.g., [8, 12, 14, 16, 21]). For GIS requirements, the main extensions to SQL include the introduction of spatial data types such as point, line and polygon, as well as spatial operators such as distance, direction, intersection and buffer. Given that spatial operators in these languages are applied in either the SELECT or WHERE clause, it becomes difficult to apply the results of spatial operators occurring in one clause (e.g., WHERE clause) to another clause (e.g., SELECT clause), and to implement further conditions on these operators such as temporality.

Different from the above approach, Gadia [11] proposed a spatial SQL in the form of "SELECT ... RESTRICTED TO · FROM ... WHERE". The condition in the WHERE clause only includes non-spatial attributes, while the condition in the augmented RESTRICTED TO clause deals with the spatial data. Huang [15] designed an extended SQL by incorporating spatial operators in the FROM clause, while the other clauses remain intact. The modified FROM clause creates a new relation with derived attributes representing the results of spatial operations. These attributes, just as other attributes in the source relations, can then be applied in both the SELECT and WHERE clauses. The direct incorporation of spatial operators, however, does not comply with the general representation of the FROM

1082-4006/97/0301~2-42$3.00

clause, as has been done by many variants of SQL.

In order to overcome this problem, this paper attempts to redesign the FROM clause via a subquery (i.e., a nested SQL), and in the meanwhile, incorporate more spatial operators including those for complicated spatial analyses such as INTERSECTION, UNION and DIFFERENCE. It should be clear that the ongoing SQL/MM [17] and SQL3 do not impose a unique form for representing spatial queries. The design of GeoSQL is conducted strictly within their framework.

The remainder of this paper is organized as follows. Section II describes the spatial data types and spatial operators in GeoSQL. The syntax and semantics of GeoSQL are discussed in Section III, with a stress on the representation of the FROM clause. Section IV gives several examples to illustrate how spatial queries are represented by GeoSQL. The interface of GeoSQL is presented in Section V, which introduces some visual constructs such as icons and ListBoxes, thereby increasing its user friendliness. Following this is an implementation of GeoSQL in Section VI. Finally, in Section VII, this paper concludes with some comments on the characteristics of GeoSQL and its future development.

II. SPATIAL DATA TYPES AND OPERATORS

Spatial Data Types

Generally, there are two kinds of spatial data models: feature-based and layer-based in GIS. The former one models spatial features while the latter one models map or a set of thematic maps [24]. The feature-based data model is currently adopted by many GIS packages such as ESRI's ArcView, MapInfo Inc.'s MapInfo and Intergraph's Modular Graphical Environment (MGE).

In a feature-based model, a spatial feature, e.g., a road, school or region, is represented as a geometric object with spatial attributes such as coordinates and topological relationships, as well as non-spatial attributes such as name, type and size. Usually, a class of features having a similar thematic property (e.g., roads, rivers or landuse) is represented by a spatial relation, and a feature corresponds to a tuple in the spatial relation. The spatial relation extends the conventional relation with an Abstract Data Type (ADT) , i.e., using GEO attribute for spatial representation. In other words, spatial attributes appear at the same conceptual level as the non-spatial attributes [22]. The basic relational operations such as projection and Cartesian product are considered applicable to spatial relations. The GEO attribute can be of point, line or polygon type.

Using the above method, the following schemas related to Hong Kong region are defined:

region (ID, name, population, GEO)
landuse (ID, type, GEO)
parcel (ID, address, GEO)
road (ID, name, class, GEO)
building (ID, name, owner, GEO)
university (ID, name, studentnum, GEO)

The features of region, landuse, parcel, building and university are of polygon type, while those of road are of line type. These tables are to be used along this paper.

Spatial Operators

Spatial operators are the methods pertaining to spatial features, which are employed to extract information from spatial features, as well as to create new spatial features from existing ones [23].

Several sets of spatial operators have been defined to query spatial database [2, 6]. Both SQL/MM and Spatial Database Engine (SDE) [9] have also defined their sets of spatial operators. Based on these, four groups of typical spatial operators are defined to illustrate how they are applied in GeoSQL.

(1) Unary spatial operators

The unary spatial operators are often used to obtain a scalar value, arcs or centroid of a spatial feature such as

ARCS(Pgn) gets the arcs from the polygon Pgn
AREA(Pgn) calculates the area of the polygon Pgn
LENGTH(L/Pgn) calculates the length of the line L or the polygon Pgn (perimeter)
CENTROID(Pgn) gets the center point of the polygon Pgn
VORONOI(Pnts) gets the VORONOI map of a pointset Pnts
BUFFER(SP) gets the buffer area of a spatial feature

Some of the above operators are type-specific, e.g., AREA, which can only take spatial features of poly-

gon type as its operands while line and point type of features are not applicable. But, some others are generic, e.g., BUFFER(SP), which can operate on one or more data types,. In this case, the spatial data type is defined as SP.

(2) Binary geometric operators

The following are the two main geometrical operators:

DISTANCE(SP1, SP2) calculates the minimum Euclidean distance between two spatial features.
DIRECTION(Pnt1, Pnt2) calculates the angle of the line connecting the points Pnt1 and Pnt2.

(3) Binary topological operators

Topological operators determine the topological relationship between two spatial features and return a Boolean value. If the topological relationship defined by an operator holds between its arguments, the operator returns the value TRUE; else FALSE.

According to [5, 6], the topological relationships usually include DISJOINT, CONTAINS, TOUCH, WITHIN, OVERLAP, CROSS, INTERSECTS and EQUALS.

(4) Binary construction operators

Construction operators may create new spatial features if a certain topological relationship holds between two spatial features. The main construction operators are:

UNION(SP1, SP2) gets all the primitive lines or polygons
INTERSECTION(SP1, SP2) gets the common part of two spatial features
DIFFERENCE(SP1, SP2) gets the different part of two spatial features

This group of operators represent the most difficult type of spatial operators to define directly in SQL [14]. However, like other types of operators, these operators in GeoSQL are represented in the way as other operators. This is discussed below.

III. REPRESENTATION OF GEOSQL

Conventional Spatial Query and Implementation Process Using GIS

When a query involves several spatial operations, we often first do spatial operations using unary spatial operators, binary spatial operators or both of them, and obtain an intermediate result. Then a selection in terms of certain criteria is carried out on this intermediate result. Finally, the desired attributes are projected. This procedure is shown in the left part of Figure 1, which provides a basis for the design of GeoSQL.

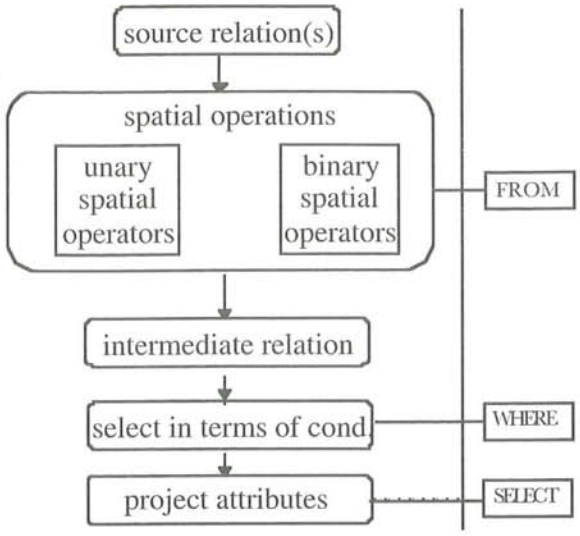

Figure 1. The conventional spatial query and implementation process using current GIS packages

Syntax of GeoSQL

Generally, a SQL statement is as follows:

SELECT $A_1, ..., A_m$
FROM $R_1, ..., R_n$
WHERE F

It is described in relational algebra as

$$\prod_{A_1,...,A_m}(\sigma_F(R_1 \times ... \times R_m))$$

Hence the FROM clause implies the Cartesian product of the given relations. In any case, the FROM clause needs to finally define a single relation, because it is the relation to which the selection in the WHERE clause and the projection in the SELECT clause are applied.

Enlightened from this, if the FROM clause is restructured to create an intermediate spatial relation resulted from spatial operations, the SQL statement can then be easily adapted to the above spatial query and implementation process (Figure 1).

An approach to implementing this is to append new attributes derived from spatial operations to the Cartesian product of source relations. Since the direct insertion of spatial operators into the FROM clause will not be consistent with the ongoing SQL3 standard, it is necessary to employ a subquery, i.e., embedding a SQL statement, in the FROM clause. The spatial operators are, therefore, applied in the nested SELECT clause for projection of their results, for example,

```
SELECT .....
FROM
    (SELECT *, CONTAINS(r. GEO, u. GEO) AS
          contval, AREA(urge) AS areaval
     FROM  region AS r, university AS u)
WHERE  .....
```

The result of CONTAINS operation is represented by the attribute "contval", and the result of AREA operation by the attribute "areaval". The subquery in the above FROM clause creates an intermediate relation, whose schema is shown in Table 1. The selection with certain conditions can then be carried out on this intermediate relation. In effect, such a relation is a virtual result because it can be optimized in terms of projection items and selection conditions during the implementation process.

This FROM clause can be described in geo-relational algebra [13] as

```
region university product
extend [contval: CONTAINS(region.GEO,
university.GEO), areaval:
          AREA(university.GEO)]
select [·]
project [·]
```

The "extend" operation above is to add new attributes to the Cartesian product of region and university.

The syntax of GeoSQL is based on the conventional SQL, whose basic form is

```
SELECT <select-clause>
FROM  <from-clause>
WHERE <where-clause>
```

Since only the FROM clause is different from that in the conventional SQL, its BNF form is described below:

```
<from-clause> :: = <relations> | <nested SQL>
<relations> :: = relation {, <relations>}
<nested SQL> :: = SELECT <sub-select clause>
    FROM <relations>
<sub-select clause> :: = *, <spatially derived at-
    tributes>
<spatially derived attributes> :: = <spatial op-
    erators> AS <attribute name> {, <spatially de-
    rived attributes> }
<spatial operators> :: = <unary spatial opera
    tors> | <binary geometric operators>
    |<binary topological operators> | <binary
    construction operators>
```

Conceptually, the subquery in the FROM clause of GeoSQL is just seen as an intermediate relation, which includes new derived attributes representing the spatial operation results. The derived attributes are taken as the same as those in the source relations, and thus can be easily applied as constraints in the WHERE clause, and referenced in the main SELECT clause for further statistical analysis or graphical display.

IV. QUERY EXAMPLES

A set of database schemas related to Hong Kong region has been defined in Section II. The following gives a group of examples to illustrate how spatial queries are represented by GeoSQL.

Example 1. Display the commercial landuse in Hong Kong.

```
SELECT  GEO
FROM    landuse
WHERE  type = 'commercial'
```

The GEO attribute occurring in the SELECT clause is to display the selected features in a map.

Example 2. Find the residences less than 2KM away from the Hong Kong Polytechnic University (HKPU), and show the distance.

Table 1. The virtual schema of the above FROM clause

Attributes of region (r)				Attributes of university(u)				New attributes	
r. ID	r. name	r. population	r. GEO	u. ID	u. name	u. studentnum	u. GEO	contval	areaval

```
SELECT  b.name, distval
FROM
   (SELECT *, DISTANCE(u.GEO, b.GEO) AS
         distval
    FROM university AS u, building ASb)
WHERE  u.name = 'HKPU' and b.type = 'resi
   dence' and  distval <= 2
```

The attribute "distval" derived in the FROM clause enables it to be applied in both the SELECT and WHERE clauses.

Example 3. Find the universities with more than 10,000 students, and indicate they are inside or outside the Kowloon region in Hong Kong.

```
SELECT  u.name, contval
FROM
   (SELECT *, CONTAINS(r.GEO, u.GEO) AS
         contval
    FROM region AS r, university AS u)
WHERE  r.name = 'Kowloon' and u.studentnum
         > 10,000
```

The derived attribute "contval" occurring in the SELECT clause shows a Boolean value indicating a selected university is inside or outside the specified region.

Example 4. Display the built-up area in Kowloon and New Territories, and sum the area.

```
SELECT  IGEO, sum(areaval)
FROM
   (SELECT *, INTERSECTION(rg.GEO,
         lu.GEO) AS IGEO, AREA(IGEO) AS
         areaval
    FROM  region AS rg, landuse AS lu)
WHERE  rg.name = 'Kowloon' or rg.name = 'New
         Territories' and lu.type = 'built-up'
```

Since landuse parcels and the regions may overlap, an intersection operation is required. Its result is specified by the spatial attribute "IGEO".

Example 5. Display the residences in Kowloon region that are nearer to HKPU than other universities.

```
SELECT  b.GEO
FROM
   (SELECT *, VORONOI(u.GEO) AS VGEO,
         INTERSECTION(VGEO, r.GEO) AS
         IGEO CONTAINS(IGEO, b.GEO) AS
         contval
    FROM  region AS r, building AS b, university
         AS u)
WHERE  r.name = 'Kowloon' and b.type = 'resi
         dence' and u.name = 'HKPU' and
         contval = TRUE
```

Voronoi operation, whose result is specified by the spatial attribute "VGEO", can meet the requirement of "nearer" function.

Example 6. Which are the roads crossing Kowloon region such that the total distance is between 30 KM and 50 KM?

```
SELECT  rd.name, rd.GEO, lval
FROM
   (SELECT *, INTERSECTION(rg.GEO,
         rd.GEO) AS IGEO,
         LENGTH(IGEO) AS lval
    FROM  region AS rg, road AS rd)
WHERE   rg.name = 'Kowloon'
GROUP BY rd.name
HAVING sum(lval) > 30 and sum(lval) < 50
```

This query shows that the result of spatial operation can also be applied to other clauses like HAVING clause besides the SELECT and WHERE clauses.

V. INTERFACE DESIGN

Although GeoSQL, an extended SQL, belongs to the textual language that is often considered incompatible with the visual language, the advantages of visual query languages such as intuitiveness and easiness [3, 19] can still be introduced in the interface design of GeoSQL to facilitate text input and reducing syntactic errors. In particular, with the development of windows programming, various visual constructs such as icons, ListBoxes and ComboBoxes can be employed in building such a user interface (Figure 2).

The interface is composed of five windows: (1) the text window, (2) the control window, (3) the ComboBoxes window, (4) the icons window, and (5)the settings window.

The text window is where the user enters the GeoSQL text following the SELECT ... FROM (SELECT - FROM) ... WHERE block. The control window contains four command buttons. The execute button passes the GeoSQL text to the implementation program. After execution, the map and attribute table are popped up to show the query result. The verify button checks the text with GeoSQL gram-

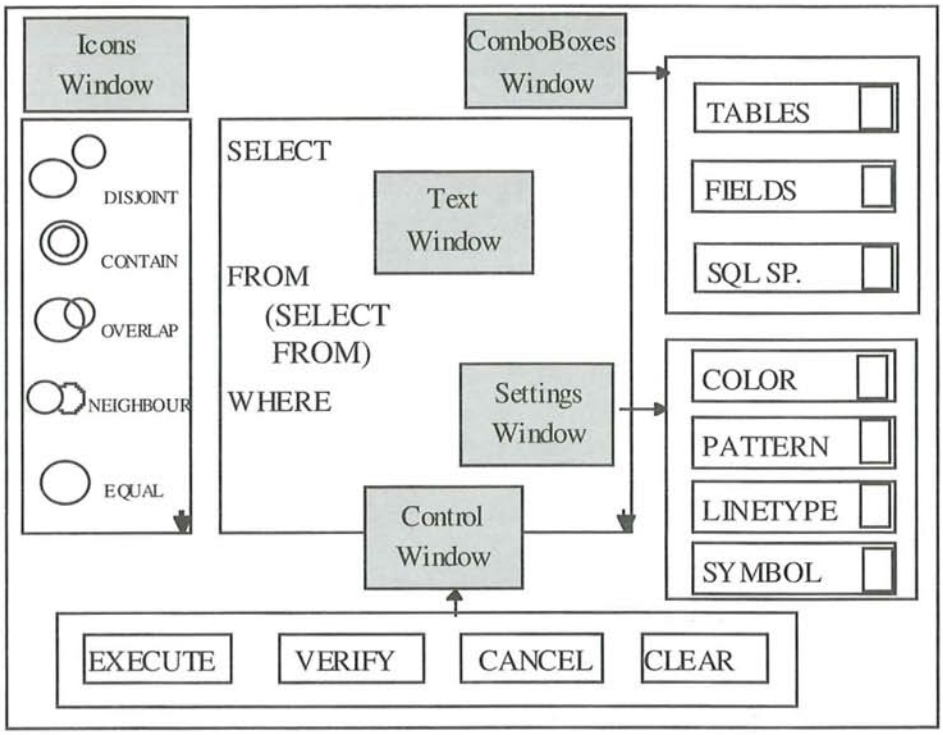

Figure 2. Window layout of GeoSQL

mar. If there is any mistake, a message is shown in a temporary pop-up window. The clear button clears the text window, and causes the already selected part of spatial objects to disappear in both the map and attribute display windows.

The ComboBoxes window contains three ComboBoxes. The "tables" stores both spatial and non-spatial tables. Once an item of the table ComboBox is selected, its corresponding fields will be added to the fields ComboBox. If two or more tables are selected, the field name will be in the form of table_name.attribute_name.

The SQL Special ComboBox lists all the predicates and operators in standard SQL. If any item in the above four ComboBoxes is selected, its corresponding text expression will occur in where the cursor locates in the text window so that the user input by keyboard is saved and the likely syntactical errors are reduced (Figure 3).
The icons window lists spatial operators in GeoSQL. The item in this ListBox differs from the ordinary ListBox in that it can combine an icon with its textual description. The ListBox is scrollable so that the user can select all the icons. It is apparent that the icons window and the ComboBoxes window are used together to assist the entry of GeoSQL text.

The settings window sets the graphical output of the query with colors, patterns, linetypes or symbols, which are realized by the four ComboBoxes respectively. The patterns for spatial data output are shown in Figure 3.

VI. AN IMPLEMENTATION

In our prototype system, GeoSQL is implemented using Oracle, Open Database Connectivity (ODBC) and Visual C++ (VC). The user interface is programmed in Visual C++ language (VC). The spatial data are stored in the Binary Large Object Block (BLOB) item in Oracle database, which are accessed through the ODBC embedded in VC programs.

After checking syntactic errors in the query text and optimization, the query processor including a spatial ODBC executes spatial operators and yields results. The spatial ODBC is composed of ODBC Application Program Interfaces (API) and APIs for implementing spatial operators. The whole implementation process corresponding to the "execute" command in the user interface is shown in Figure

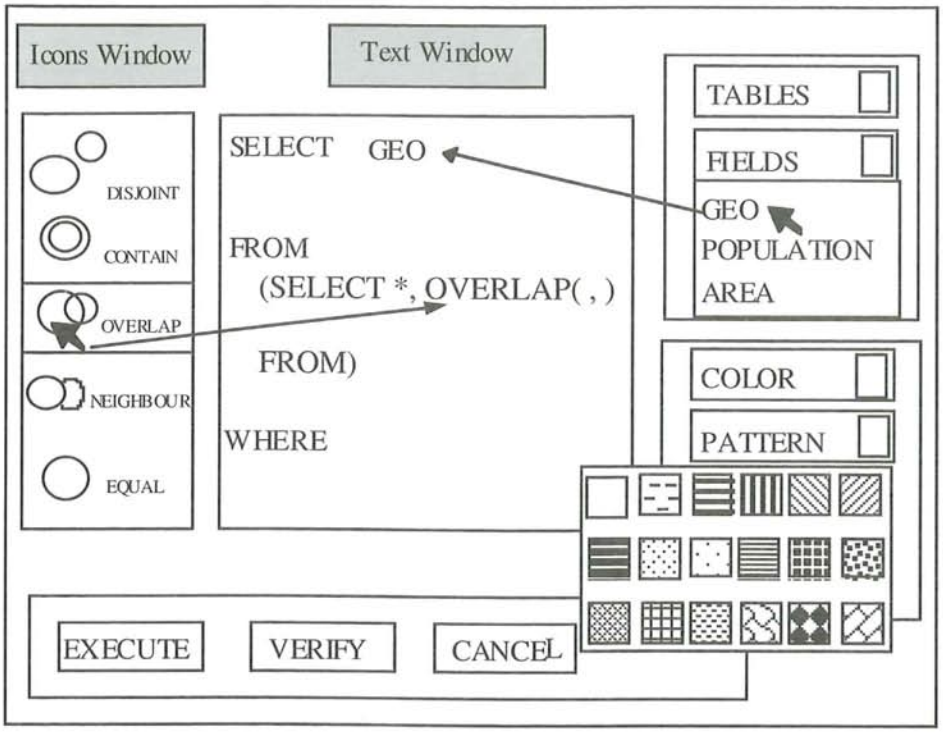

Figure 3. The assistance of visual constructs for GeoSQL text entry and the settings for graphical output

4. Using such a method, the example 4 in Section IV is implemented and the result is shown in Figure 5.

VII. CONCLUSIONS

This paper describes a different approach to designing an extended SQL in terms of the conventional spatial query and implementation process. By incorporating spatial operators in the FROM clause via a subquery, GeoSQL is well adapted to the conventional SQL design principles. More importantly, it becomes possible to apply the results of spatial operations in the SELECT, WHERE and HAVING clauses. Because the result of a spatial operation in GeoSQL is treated as a derived attribute, the further conditions such as temporality (e.g., valid-time) and fuzziness (e.g. "very far" and "overlap significantly") can be acted on it. In this sense, GeoSQL holds promise in expressing the spatio-temporal and fuzzy queries.

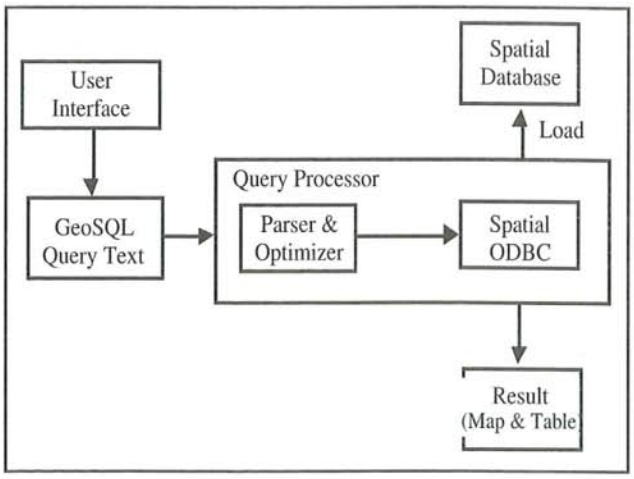

Figure 4. An implementation of GeoSQL

The interface design of GeoSQL introduces a set of visual constructs, which aid the entry of query text and reduce the possible syntactical errors. Such an approach overcomes the problems of previous extended SQLs, which compose a query only by pure text input.

The implementation of GeoSQL is a non-trivial task. Currently only part of the spatial operators can be implemented. Since SDE is now available in our laboratory, the spatial operators are to be realized

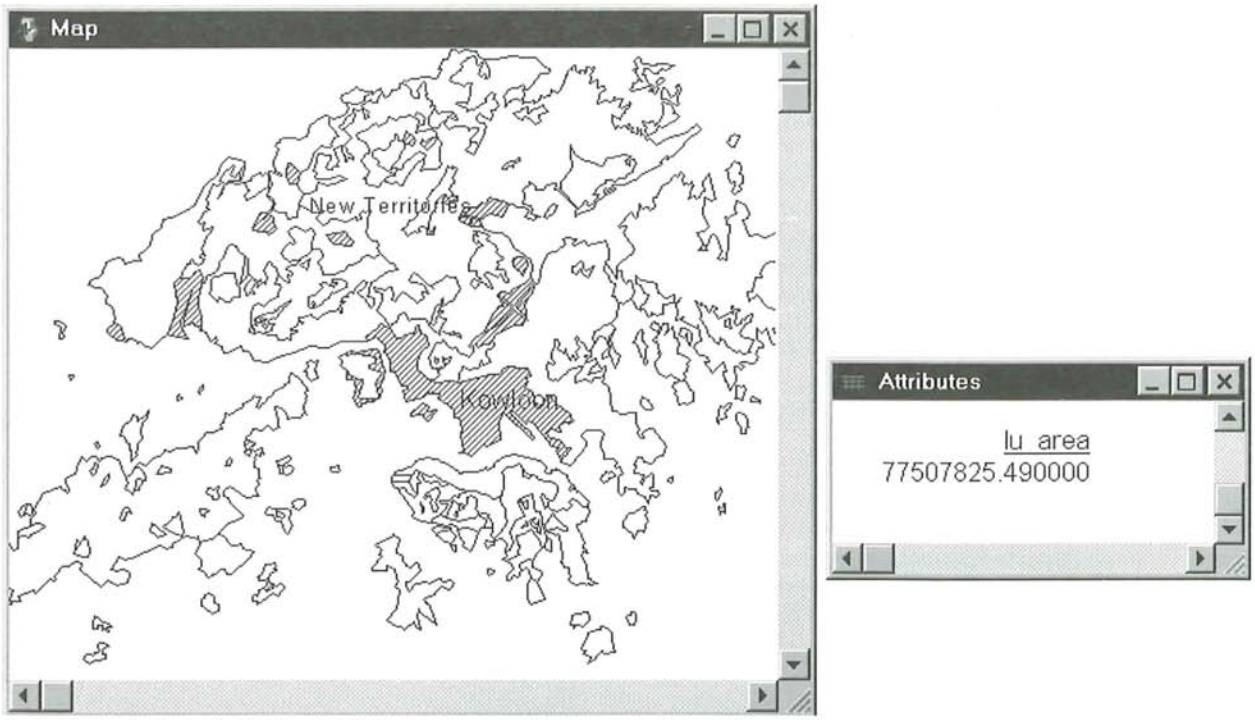

Figure 5. Experimental result of example 4 in Section IV

by it. The connection of SDE with our programs still needs to be done. In addition, more attention is also needed to the optimization of the language during its implementation.

ACKNOWLEDGMENTS

This research is partially supported by the Research Grants Council of HKSAR government under RGC earmarked research grant No. CUHK 150/96H, CUHK RAC under grant No. 4720401, and the National Natural Science Foundation (NNSF) of China under grant No. 49501013. We are grateful to Professors Guanhua XU and Shouyong Yan for their thoughtful suggestions at the early stage of this research.

REFERENCES

[1] American National Standards Institute (ANSI), 1989, *X3.135-1989 Database Language SQL*.

[2] Boursier, P. and M. Mainguenaud, 1992, Spatial query languages: extended SQL vs. visual language vs. hypermaps. *5th International Symposium on Spatial Data Handling*, Charleston, SC, USA.

[3] Calcinelli, D. and M. Mainguenaud, 1992, Cigales: a visual query language for Geographical Information System: the user interface. *Journal of Visual Languages and Computing*, 5, 113-132.

[4] Chu, T.H. and Y.T. Lung, 1995, Design a spatial language applying behavioral approach. In *Proceedings of GeoInformatics' 95*, Hong Kong, pp. 851-867.

[5] Clementini, E., D. Paolino and P. Oosterom, 1993, A small set of formal topological relationships suitable for end-user interaction. In D.J. Abel and B.C. Ooi (eds.), *Advances in Spatial Databases*, SSD'1993, Singapore, pp. 277-295.

[6] Egenhofer, M. and R. Franzosa, 1991, Point-set topological spatial relations. *International Journal of Geographical Information Systems*, 5(2): 161-174.

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

[8] Egenhofer, M., 1994, Spatial SQL: a query and presentation language. *IEEE Transactions on Knowledge Engineering and Data Engineering*. 6(1): 86-95.

[9] ESRI, 1996, *SDE Developer's Guide Version 2.1*, Environmental Systems Research Institute, Inc.

[10] Frank, A., 1982, Mapquery - Database query languages for retrieval of geometric data and its graphical representation. *ACM Computer Graphics*, 16(3): 199-207.

[11] Gadia, S.K., 1993, Parametric databases: seamless integration of spatial, temporal, belief, and ordinary data. *SIGMOD Record*, 22(1): 15-20.

[12] Goh, P.C., 1989, A graphic query language for cartographic and land information systems. *International Journal of Geographical Information Systems*, 3(1):

245-255.

[13] Guting, R.H., 1988, Geo-relational algebra: a model and query language for geometric database system. In J.W. Schmidt, S. Ceri and M. Missikoff (eds.), *Proceedings of the International Conference on EDBT*, Venice, pp. 506-527.

[14] Herring, J.R., R.C. Larsen and J. Shivakumar, 1988, Extensions to the SQL query language to support spatial analysis in a topological database. In *Proceedings of GIS/LIS' 88*, San Antonio, 30 November-2 December.

[15] Huang, B., 1997, GeoSQL: a Visual Spatial SQL for Topological Relationships in GIS. In *Proceedings of the Workshop on Dynamic and Multi-dimensional GIS*, The Polytechnic University of Hong Kong, Hong Kong, August.

[16] Ingram, K.J. and W.W. Phillips, 1987, Geographic information processing using a SQL based query language. In *Proceedings of AutoCarto 8*, Baltimore, Maryland, pp. 326-335.

[17] ISO, 1995, *SQL Multimedia and Application Packages (SQL/MM) Part3: Spatial*, ISO Working Draft, September.

[18] Kruglinski, D.J., 1993, *Inside Visual C++*. Microsoft Press, USA.

[19] Lee, Y.C. and F.L. Chin, 1995, An iconic query language for topological relationships in GIS. *International Journal of Geographical Information Systems*, 9(1): 25-46.

[20] Ooi, B.C., 1988, Efficient query processing for Geographic Information System. PhD thesis, Monash University, Victoria, Australia.

[21] Roussopoulos, N., C. Faloutsos and T. Sellis, 1988, An efficient pictorial database system for PSQL. *IEEE Transactions on Software Engineering*, 14(5): 639-650.

[22] Samet, H. and W. Aref, 1994, Spatial data models and query processing. In W. Kim (ed.), *Modern Database Systems: The Object Model, Interoperability, and Beyond,* Addison Wesley/ACM Press, Reading, MA.

[23] Svensson, P. and Z.X. Huang, 1991, A query language for spatial data analysis. In *2nd Symposium on Large Spatial Databases*, Zurich, Switzerland.

[24] Tang, A.Y., T.M. Adams and E.L. Usery, 1996, A spatial data model design for feature-based Geographical Information Systems. *International Journal of Geographical Information Systems*, 10(5): 643-659.

在五周年庆祝会上，CPGIS 成员与中国科学院遥感所部分领导合影。从左到右：林珲、宫鹏、郭华东(遥感所所长)、孙以义(CPGIS 上海站经理)、丁跃民、柳林、李斌、周启鸣、杨崇俊(中科院百人计划入选者)、王超(遥感所副所长)、沈小平、周成虎(地理所资源与环境信息系统国家重点实验室主任) 和关蔚禾。