Solving Spatial Analysis Problems with GeoSAL,  
A Spatial Query Language

Zhexue Huang  
Department of Photogrammetry  
Royal Institute of Technology  
S - 100 44 Stockholm, Sweden

Per Svensson  
Division of Applied Mathematics and Data Processing  
National Defence Research Establishment  
S - 172 90 Sundbyberg, Sweden

Hans Hauska  
Department of Photogrammetry  
Royal Institute of Technology  
S - 100 44 Stockholm, Sweden

Abstract

Database query languages are commonly used as a tool to retrieve data of interest from databases. Although some query languages provide basic functions useful for statistical data analysis and others contain spatial query capabilities, there is as yet no query language which satisfies even basic requirements of spatial analysis.

By including a set of fundamental spatial operators into the query language of an extensible DBMS, a number of typical spatial operations in GISs can, however, be formulated as queries in the language. Hence, in the future spatial analysis tasks could be carried out by using such a query language, relieving users from procedural programming while retaining the advantages of a systematic language structure embedding fundamental as well as domain-specific concepts.

This paper shows how to perform spatial queries and analyses with GeoSAL, a spatial query language designed for an extended relational DBMS, which is being developed at the National Defence Research Establishment in Stockholm. GeoSAL is based on an extended relational data model which integrates the object and layer models commonly found in GISs. Spatial data types and operators are part of the language. Examples given in this paper demonstrate the use of GeoSAL in modeling typical spatial operations and performing spatial analysis.

1. Introduction

Database query languages are commonly used as a tool to retrieve information from databases. Most query languages currently used were designed for this purpose and very few for data analysis [Tans91, Kara85].

In recent years substantial efforts have been made to extend conventional query languages [Egen89, Rous88, Lori91, Aref91] or to design new ones [Guti88, Main90, Goh89] to suit spatial applications.

Although some query languages provide basic functions for statistical calculations, like average, standard deviation, minimum, maximum, etc., and others contain capabilities for spatial querying, there is as yet no query language implementation which satisfies even basic requirements of spatial analysis tasks. Such tasks usually have to be solved with the aid of application-specific modules of a geographical information system (GIS).
GIS as tools for spatial analysis have been widely used in different applications. Some GIS products have reached an impressive level of functionality. Thus, in many cases the major obstacle to the use of GIS in spatial analysis applications is not lack of functionality but the large number of concepts and commands that have to be mastered [Good90a].

An effort has been made by Tomlin [Toml90] to structure spatial analysis functions for a large number of cartographic modeling tasks into a limited set of primitive operations. Tomlin's work is a pioneering contribution to the design of self-contained spatial analysis languages. The fact that it is based on a simple raster data structure rather than on an elaborate type structure as required in large spatial databases, however, prevents the direct introduction of Tomlin's operator structure into spatial database languages.

Another obstacle to the use of GIS is the lack of a generic data model for different applications. Some GIS products are interfaced with conventional DBMSs to support data management, while the analysis of spatial data is performed in the GIS subsystem. One problem in these systems is the conflict between the complexity of spatial objects in GISs and the simple type structure (data model) offered by contemporary DBMSs, which typically support only basic data types and a simple record structure [Haas91].

Research on extensible database management systems (EDBMS) [Haas91] and on object-oriented database management systems (OODBMS) [Unla90] aims at providing generic models for non-standard applications. At its present stage, work on such systems is concentrated on data handling and simple ad hoc spatial queries. Again, the data analysis capability of their query languages is quite limited [Haas91].

By including a set of fundamental spatial operators into the query language of an extensible DBMS, typical spatial analysis operations in GISs can, however, be performed as data transformations, dressed as queries in the language. Hence, in the future spatial analysis tasks could be carried out by using such a query language, relieving users from procedural programming while retaining the advantages of a systematic language structure embedding fundamental as well as domain-specific concepts.

In this paper we show how to perform spatial queries and analyses with GeoSAL [Sven91], a spatial query language based on an extended relational data model which integrates the object and layer models commonly found in GISs. GeoSAL is designed for a prototype spatial analysis and decision support system which is being developed as an extension of the relational data analysis system Cantor, designed and built by the National Defence Research Establishment (FOA) in Stockholm [Kar83,86]. Throughout this paper, GeoSAL concepts introduced in [Sven91] will be used without further reference.

This work was performed under contract with a joint project between FOA, NobelTech Systems AB, and Ericsson Radio Systems AB. This project is part of a national R & D program in Information Technology.

The paper is organized as follows. In Section 2, classes of operations required in spatial analysis are reviewed. These operations can be performed by contemporary GIS software, but only a few are discussed in the above-mentioned proposals for spatial query languages. The use of the type system of GeoSAL in defining spatial data types and relations in Cantor is demonstrated in Section 3. The operations reclassification, generalization and overlay are expressed in Section 4 as sequences of queries in GeoSAL. It is also shown in Section 4 how high level operations are generalized from a sequence of query statements. The partitioning of the solutions into queries is a matter of convenience. Equivalently, each solution can be written as a single expression. In Section 5, simple examples are given which show how to use GeoSAL to perform spatial queries and solve spatial analysis problems. Conclusions are drawn in Section 6.

2. Classes of operations required for spatial analysis

A study on what kind of operations are required in spatial analysis was carried out before the specifications of a spatial database language was outlined. One conclusion of this study was that basic operations required in spatial data analysis could be arranged in the following six categories:
(1) data selection and transformation,
(2) reclassification and generalization,
(3) "measurement",
(4) neighborhood,
(5) overlay,
(6) statistics.

Operations in each of these categories can be performed by contemporary GIS software, but only the most basic can be expressed in the spatial query languages previously proposed.

In GeoSAL, five classes of spatial operators are defined. They are:

(a) Unary geometrical operators which extract geometrical data from one object.
(b) Unary object transformation operators that transform one spatial object into another.
(c) Binary geometrical operators which compute geometrical relationships between two or more objects.
(d) Binary topological operators which test topological relationships of two objects.
(e) Object construction operators which construct new objects from existing objects.

There is no one-to-one correspondence between the six categories defined above and the classes of spatial operators in GeoSAL. However, a subset of the spatial analyses in these categories can be performed using either a single GeoSAL operator or a sequence of operators in one or more query expressions.

In the remainder of this paper, we show how to carry out spatial analysis operations in the categories (2) and (5) using query language expressions which involve operators mainly from the classes (d) and (e).

3. The type system of GeoSAL

Types in GeoSAL form an inheritance structure [Grap92]. Operators are defined on these data types. Some of them are polymorphic, i.e., change their algorithmic behaviour automatically as required by the current representation of their operands.

A type which inherits from another is called a specialization of the latter, and contains all operators and other features of its ancestor. System-defined specializations may possess features in addition to those of its ancestors.

A facility allowing users to define their own data types is also provided. To allow the use of abstract system-defined concepts in such data types, the notion of generic type is frequently required. A generic type is only partially specified and must be supplied with additional information to form a specific type definition. In GeoSAL, this information is supplied as type parameters.

When modeling spatial information, two fundamentally different data representations, usually called vector (or geometric) and raster (or image) representation, are frequently used. In certain types of analysis such as differential analytic computations over a surface, raster representation is required, whereas vector representation could be more convenient for example for network analysis.

In GeoSAL, a spatial data type, such as Point, Line, or Polygon, defines an abstract data structure as a specialization of a tuple or relation type, and is used to model spatial objects in vector representation.
Also, in spatial modeling two views, object and layer, are commonly used [Good90b]. A layer is a set of spatial objects with the same spatial type and category attributes.

In GeoSAL these two views are integrated in the sense that spatial objects can be organized into layers based on their spatial types and non-spatial category attributes. GeoSAL provides the generic type Tessellation for layers of polygons in vector representation (Section 3.1). For the representation of raster layers, the generic type Raster is provided. Raster (Section 3.2) is a specialization of RegularTessellation which is a specialization of Tessellation.

The syntax for defining spatial database objects follows the pattern:

```
DEFINE OBJECT SObjectSet: SpatialObjectSet;
```

where the type SpatialObjectSet is defined as:

```
DEFINE TYPE SpatialObjectSet := SET_OF SpatialObject;

DEFINE TYPE SpatialObject := .(id::STYPE, attr1::BTYPE, attr2::BTYPE,...);
```

where:

- SpatialObject - a structured tuple type, modeling spatial objects
- SpatialObjectSet - a spatial object set type
- SObjectSet - a spatial object set
- id - object identifier
- STYPE - system-defined spatial types such as Point, Line, and Polygon
- BTYPE - basic data types such as Integer, Float, Literal

The double colons of id::STYPE and attr1::BTYPE denote the key attributes of the tuple type SpatialObject.

The semantic properties of the GeoSAL data model can be summarized as follows:

1. Every database object has a well-defined, named type. Type names may be used in other type and object definitions.

2. Attributes and tuples may be structured. An attribute type is a pair <name> : <value type>, where <value type> can be any GeoSAL type expression.

3. The instances of a given tuple type are individual database objects, i.e., they have a name, unique within their contextual scope, may have a defining view and may possess a value. Tuple operations can be used to identify, aggregate, and disaggregate tuples.

4. Set types are sets of scalars of the same type ("scalar sets") or sets of tuples of the same type ("relations"). Set operations may be applied to expressions of any set type, relation operations only to expressions of relation type.

5. Although a tuple instance is an individual object, it may contain subtuples whose values are relations.

6. The notion of key applies to both tuple and relation types. A key is a subtuple whose value must be unique for each tuple instance in a relation. The key propagation semantics is well-defined for each tuple, set, and relation operator. In SpatialObjectSet, the attributes id and attr1 form a composite (multi-dimensional) key. By restricting SObjectSet to a specific attr1 value, a set-valued, spatially non-contiguous "layer object" is obtained.
3.1 Vector data types

A spatial object set type in vector representation can be defined as follows:

```
DEFINE OBJECT PolygonalSet: PolygonalObjectSet;
```

assuming that the following types are defined:

```
DEFINE TYPE PolygonalObjectSet := SET_OF PolygonalObject;
DEFINE TYPE PolygonalObject := .(pg::Polygon, attr1:BTYPE, attr2:BTYPE,...);
```

PolygonalObject has an attribute of the system-provided spatial type Polygon, which is structurally equivalent to:

```
DEFINE TYPE Polygon := .(id::Integer, vertices:SET_OF Point);
```

The structure of a Point object is equivalent to:

```
DEFINE TYPE Point := .(id::Integer, xc:Float, yc:Float);
```

Here, id represents identifiers of the points which form the polygon boundary, and xc and yc the coordinates of these points.

To explain the relationship between a spatial and a non-spatial data type, we compare the set object PolygonalSet defined above with a similar non-spatial object:

```
DEFINE OBJECT NonSpatialSet: NSObjectSet;
```

assuming that the following types are defined:

```
DEFINE TYPE NSObjectSet := SET_OF NSObject;
DEFINE TYPE NSObject := .(nso::NSItems, attr1:BTYPE, attr2:BTYPE,...);
DEFINE TYPE NSItems := SET_OF .(id::Integer, xc:Float, yc:Float);
```

The same non-spatial operations can be applied to the database objects PolygonalSet and NonSpatialSet, whereas spatial operators are applicable only to PolygonalSet and its members. Only simply connected polygons, possibly with simply connected polygonal holes, are allowed as instances of type Polygon. Similar constraints are defined for most system-provided spatial types. For the remainder of this paper, the term polygon is used in the sense "instance of type Polygon". When we want to refer to planar regions in general, the term region will be used.

The types PolygonalObjectSet and NSObjectSet are said to be structurally conformant. To avoid ambiguities, type conversion must be explicitly specified whenever two objects of structurally conformant types are used as operands of a polymorphic operator, such as UNION.

By definition, the geometric aspect of a tessellation is a set of edge-adjacent or disjoint polygonal regions. The complete tessellation is formed by attaching non-spatial categorical data to the regions. To accomplish this, a type parametrization mechanism is applied. The system-provided parameterized ("generic") data type Tessellation is structurally equivalent to:

```
DEFINE TYPE Tessellation [T -> Tuple] := SET_OF .(pg::Polygon, attr:T);
```

For example,

```
DEFINE OBJECT Landcover: Tessellation [coverage:Literal];
```
defines a layer with the single category attribute coverage. The key attribute of this relation has the type pg::Polygon. Operations on instances of type Layer may use semantic knowledge which is not available for general sets of polygons. We discuss this further in Section 4.1.

The set of regions which form a tessellation is not necessarily minimal with respect to the category attributes, i.e., two adjacent regions may have the same category value.

3.2 Raster data types

A set of raster layers with the same regular tessellation can be represented as a relation, for example:

\[
\text{Images}(ix::\text{Integer}, iy::\text{Integer}, image1::\text{Integer}, image2::\text{Float});
\]

It has been shown [Stje86] that many spatial analysis tasks can be expressed under this representation, using the algebraic query language SAL [Arnb80]. However, the lack of spatial knowledge in this representation frequently prevents efficient evaluation of queries.

Using spatial concepts of GeoSAL, the above relation Images can be redefined as:

\[
\text{DEFINE \ OBJECT \ PixelImages: ImageLayer;}
\]

where ImageLayer is defined by:

\[
\text{DEFINE \ TYPE \ ImageLayer := XY_Raster [ImageElement]}
\]

\[
\text{WITH \ [.ll = OSquare, .m = nx, .n = ny];}
\]

\[
\text{DEFINE \ TYPE \ ImageElement := \.(ix::\text{Integer}, iy::\text{Integer}, image1::\text{Integer}, image2::\text{Literal});}
\]

Here, the parameter ImageElement specifies the key and category attributes of the relation PixelImages. The restriction clause, which follows the keyword WITH, is used to specify values for the geometric parameters .ll, .m, and .n (. is a short form for a semantically redundant prefix sequence of attribute names). The arguments nx and ny denote the number of image elements along each principal direction of the image area.

Furthermore, the parameter .ll, i.e., the "lower left" raster element of the image area must be specified, in this case by being instantiated by an object of the system-defined type XYSquare:

\[
\text{DEFINE \ OBJECT \ OSquare: XYSquare := <some expression> ;}
\]

The relation PixelImages represents two layers, each partitioning the space into a minimal set of regions, homogeneous with respect to its category attribute value. The regions in each layer are not individually identified.

The notion of spatial object is meaningful irrespective of whether vector or raster representation is used. In GeoSAL, raster data can therefore also be represented with region (or object) identification, where raster elements belonging to the same homogeneous region are assigned a common identifier.

Region identification is performed by the operator OBJECTIFY, used as in the example:

\[
\text{PixelRegions := PixelImages [regid:OBJECTIFY(image1)];}
\]

In this expression, an additional key attribute regid is added to the image1 layer, forming a new relation PixelRegions with an attribute regid which identifies the homogeneous regions of the layer.
4. Performing spatial operations with GeoSAL

The spatial operations listed in Section 2 can be freely combined into query expressions, or "views," in GeoSAL. In this section we show how to perform reclassification, generalization and overlay operations using a sequence of views in GeoSAL. These three operations are frequently used in spatial analysis. Their expression in GeoSAL depends critically on the syntax and semantics of the four spatial operators MEETS, UNION, DIFFERENCE, and INTERSECTION, which is described in Section 4.1.

4.1 Syntax and semantics of the spatial operators MEETS, UNION, DIFFERENCE and INTERSECTION

MEETS is a binary logical operator which checks a topological relationship between two polygons. MEETS is true if and only if the two polygons share a boundary segment or a vertex, and their interiors have no common point. For members a, b of a tessellation, (a MEETS b) is equivalent to NOT(a DISJOINT b).

Example:

If LR1 is a tessellation with the category attribute attr1 (Figure 4.4), the query expression

\[
\text{LR2:Tessellation[attr1:Literal] := } *(a:LR1, b:LR1) \\
\text{WHERE [(a.attr1=b.attr1) AND (a.pg MEETS b.pg)]} \\
\text{[pg::a.pg, attr1::a.attr1]};
\]

of Section 4.3.1 produces a new layer LR2, consisting of the regions p2, p3, p4 and p7 in Figure 4.4. This set of homogeneous regions is not minimal.

In this query, *(a:LR1, b:LR1) is the Cartesian product of the two relations a:LR1 and b:LR1. The expression a:LR1 forms a relation whose attributes are the same as those of LR1 except that their names are prefixed with "a.". The second expression within brackets is a "generalized projection" in which names and values of the result attributes pg and attr1 are defined.

The polymorphic operators UNION and INTERSECTION can be used in two syntactic contexts: as binary operators and as aggregation operators, analogous to SUM and PRODUCT in a non-spatial query language.

Binary UNION maps pairs of polygons, and pairs of polygon sets, to sets of polygons. Let X and Y be point sets representing polygons and let p be an arbitrary point. The result point set of the binary UNION operator is then defined by:

\[
X \cup Y = \{ p \mid (p \in X) \lor (p \in Y) \}
\]

If the two polygons are 1-disjoint, i.e., disjoint with the exception of isolated points, the trivial result is the set \(X, Y\). Otherwise, the result is the singleton polygon set whose member coincides with the result point set. Note that the union of two hole-free polygons is not hole-free in general.

Like other aggregation operators, aggregation UNION can only be applied to relation attributes. Furthermore, only set-valued attributes are meaningful operands.

If pg is a polygon attribute in the relation R, the expression R UNION [pg] forms the set of 1-disjoint polygons whose points coincide with the point set \(\{ p \mid (p \in F1) \lor \ldots \lor (p \in Fn) \}\), where F1, ..., Fn are the members of R [pg]. We will say that the result is the minimal polygonal cover of the points in the pg polygons.

Partitioned aggregation (denoted by the "group by" clause in some query languages) with respect to a category P can be expressed as R [pg:: UNION(pg), P]. Here, P is a possibly multidimensional category attribute in R, partitioning the set of pg polygons into distinct "P-classes", one for each value of P. The polygons may be spatially overlapping. The value of the expression will be equal to that obtained by applying UNION to each P-class of polygons separately. The result polygons in each P-class are assigned unique pg identifiers.
When applied to the polygon attribute of a tessellation, the semantics of UNION is simplified: whenever the relationship MEETS holds between polygons associated with the same category attribute value, the UNION is formed of all the polygons in the transitive closure of the relationship. Each distinct polygon in the result is assigned a unique identifier.

Example:

The expression

\[ \text{LR2} [\text{pg::UNION(pg), attr1}] \]

where LR2 is a tessellation with the category attribute attr1, is evaluated as follows:

For each distinct class value of the category attribute attr1, the union is formed of all polygons in this class which meet. Isolated polygons are left unchanged. Finally, unique identifiers pg are assigned to each resulting region in the class.

Figure 4.1 shows the result of applying UNION as a partitioned aggregation operator over a tessellation.

The semantics of INTERSECTION is analogous to that of UNION, substituting OR with AND in the definitions. INTERSECTION over a tessellation, however, produces a "degenerate" result, i.e., the boundary lines between adjacent polygons.

The operator DIFFERENCE can be used as a binary operator which maps a pair of polygons, or a pair of sets of disjoint polygons, to a set of polygons. Unlike UNION and INTERSECTION, DIFFERENCE can not be given a meaningful interpretation as an aggregation operator.

The result point set of the binary DIFFERENCE operator is defined by:

\[ X \text{DIFFERENCE} Y = \{ p \mid (p \text{ in } X) \text{ AND } (p \text{ not in } Y) \} \]

The result is the minimal polygonal cover of this point set. If \( X, Y \) do not intersect, DIFFERENCE copies \( X \).

When the first operand, \( X \), is the polygon attribute of a tessellation with a category attribute \( P \), and the second, \( Y \), is a constant set of disjoint polygons, the semantics of \( R [X \text{DIFFERENCE} Y, P] \) is the following:

For each \( P \)-class of the tessellation, the minimal polygonal cover of the set of polygons, resulting from successive application of \( xi \) DIFFERENCE \( Y \) to each member \( xi \) of the \( P \)-class, is formed. Each distinct polygon in the result is assigned a unique identifier.

Figure 4.2 visualizes the binary set operations DIFFERENCE and INTERSECTION applied to polygons.

Example:

Let SR1 and SR2 be tessellations with the attributes pg::Polygon and attr:Literal.

The expression

\[ \text{SR1 UNION SR2} \]
forms the minimal polygonal cover of the polygons in SR2.

The expression

\[ \text{SR1} \mid \text{pg \ DIFFERENCE } (\text{SR2 UNION}[\text{pg}]), \text{attr1} \]

produces a set of 1-disjoint polygons for each value of attr1. These polygons are assigned different identifiers in the result relation. If no two polygons in SR1 and SR2 intersect, the expression reproduces SR1.

### 4.2 Reclassification

Figure 4.3 (a) shows a layer consisting of several subregions, defined by distinct "class" values of a category attribute. Four different class values, A, B, C and D, represent four groups of regions (layer objects) of the same class. The task is to reclassify the regions of class C to B.

![Diagram of layer LR with subregions and class values](image)

**Figure 4.3.** (a) The layer LR with subregions p1, p2, ... and class values A, B, ...

<table>
<thead>
<tr>
<th>Name: LR</th>
</tr>
</thead>
<tbody>
<tr>
<td>pg</td>
</tr>
<tr>
<td>-----</td>
</tr>
<tr>
<td>p1</td>
</tr>
<tr>
<td>p2</td>
</tr>
<tr>
<td>p3</td>
</tr>
<tr>
<td>p4</td>
</tr>
<tr>
<td>p5</td>
</tr>
<tr>
<td>p6</td>
</tr>
<tr>
<td>p7</td>
</tr>
<tr>
<td>p8</td>
</tr>
<tr>
<td>p9</td>
</tr>
</tbody>
</table>

(b) The table representation of the layer.
We represent each region as a polygon and the layer as a tessellation relation as follows:

```
DEFINE OBJECT LR: Tessellation [attr1:Literal];
```

Hiding the geometric data from view, the representation of the relation LR can be visualized as the table of Figure 4.3 (b) in which pg is an object identifier whose value is unique within the relation.

The query:

```
LR1:Tessellation[attr1:Literal] := LR [pg, attr1:IF attr1 = 'C' THEN 'B' ELSE attr1];
```

reclassifies the regions of group C to B. The semantics of the query is the following: if the value of attribute attr1 in LR is C, then assign B to the value of the same attribute in LR1, otherwise, move the value in LR to LR1. The values of attribute pg are unchanged.

The layer LR1 is visualized in Figure 4.4.

![Figure 4.4. The layer of Figure 4.3 after the substitution B <- C.](image)

Alternatively, we can define a function to assign the new category values. For example, the above query can be written as

```
Reclassify(class:Literal):Literal := IF class = 'C' THEN 'B' ELSE class;

LR1:Tessellation[class:Literal] := LR [pg, class:Reclassify(class)];
```

It is also possible to define the reclassification transformation as a relation table. This method is preferable for complex transformations.

### 4.3 Generalization

Generalization is a spatial operation which merges adjacent regions having the same category values [Tom81]. For example, the reclassified layer in Figure 4.4 contains adjacent regions with the same category value B. The common boundaries between these regions must be removed (see Figure 4.5). In GeoSAL this operation can be carried out by the object construction operator UNION.

#### 4.3.1 Performing the generalization operation by queries

Assume that we want to apply generalization to the result LR1 of Section 4.2. The generalization operation can be performed by the following sequence of expressions:

1. Find regions which meet others with the same category attribute value.

```
LR2:Tessellation[attr1:Literal] := *(a:LR1, b:LR1)
WHERE [(a.attr1 = b.attr1) AND (a.pg MEETS b.pg)]
[pg::a.pg. attr1:a.attr1];
```
(2): Find regions which do not meet others with the same attribute value.

\[ \text{LR3:Tessellation[attr1:Literal]} := \text{LR1 DIFFERENCE LR2}; \]

(3): Merge regions which have the same attribute value and meet.

\[ \text{LR4:Tessellation[attr1:Literal]} := \text{LR2 [pg::UNION(pg), attr1]}; \]

(4): Combine LR3 and LR4:

\[ \text{LR5:Tessellation[attr1:Literal]} := \text{LR3 UNION LR4}; \]

LR5 is visualized in Figure 4.5 (a) and can also be described by the table of Figure 4.5 (b).

![Diagram](a)

<table>
<thead>
<tr>
<th>Name: LR5</th>
</tr>
</thead>
<tbody>
<tr>
<td>pg</td>
</tr>
<tr>
<td>-----</td>
</tr>
<tr>
<td>p1</td>
</tr>
<tr>
<td>p2</td>
</tr>
<tr>
<td>p3</td>
</tr>
<tr>
<td>p4</td>
</tr>
<tr>
<td>p5</td>
</tr>
<tr>
<td>p6</td>
</tr>
</tbody>
</table>

(a) Figure 4.5. (a) The layer generalized from the layer of Figure 4.4.
(b) The table representation of the layer.

4.3.2 Formulating the generalization operation as a function

Since the generalization operation is a somewhat complex procedure which is also frequently used, it is worthwhile to express the above queries as a function which is able to perform various generalization tasks. This function can be written as the following sequence of queries:

\[ \text{GEN1(LR:Tessellation [attr -> Scalar]):Tessellation[attr -> Scalar]} := \]
\[ *(a:LR, b:LR) \text{ WHERE [(a.attr=b.attr) AND (a.pg MEETS b.pg)]} \]
\[ \text{[pg::a.pg, attr::a.attr]}; \]

\[ \text{GEN2(LR:Tessellation [attr -> Scalar]):Tessellation[attr -> Scalar]} := \]
\[ \text{LR DIFFERENCE GEN1(LR)}; \]

\[ \text{GEN3(LR:Tessellation [attr -> Scalar]):Tessellation[attr -> Scalar]} := \]
\[ \text{GEN1(LR) [pg::UNION(pg), attr]}; \]

\[ \text{GENERALIZE(LR:Tessellation [attr:Scalar]):Tessellation[attr -> Scalar]} := \]
\[ \text{GEN2(LR) UNION GEN3(LR)}; \]

Using GENERALIZE, the above example can be performed as:

\[ \text{LR6:Tessellation[attr1:Literal]} := \text{GENERALIZE(LR1)}; \]
4.4 Overlay

The overlay operation, one of the most frequently used spatial operations in a GIS, refers to the process of generating new layers from existing layers registered to a common space. The input to the overlay operation is a set of layers, and the output is a new layer in the same space [Toml90]. In GeoSAL this operation is carried out by using the spatial operators DIFFERENCE, INTERSECTION and UNION, as described below.

Figure 4.6 shows two layers SR1 and SR2. We want to overlay them to generate a new layer.

![Figure 4.6. Two layers (tessellations).](image)

The two layers are represented as follows:

```
DEFINE TYPE Layer := Tessellation [attr1:Literal];
DEFINE OBJECT SR1,SR2: Layer;
```

The result is obtained by the following sequence of expressions:

1. Difference between the tessellation SR1 and the union of SR2 (Figure 4.7):
   ```
   LSR1:Layer := SR1 [pg DIFFERENCE (SR2 UNION[pg]), attr1];
   ```

2. Difference between the tessellation SR2 and the union of SR1 (Figure 4.8):
   ```
   LSR2:Layer := SR2 [pg DIFFERENCE (SR1 UNION[pg]), attr1];
   ```

3. Intersection of the two tessellations (Figure 4.9):
   ```
   LSR3:Tessellation[attr1:Literal, attr2:Literall := *(a:SR1, b:SR2)
   [pg:=(a.pg INTERSECTION b.pg), attr1:a.attr1, attr2:b.attr1];
   ```

4. Reclassification of LSR3:
   ```
   LSR4:Layer := LSR3 [pg, attr1:IF (attr1=...) AND (attr2=...) THEN ... ELSE ...];
   ```
5. Spatial query and analysis examples

5.1 Query examples

Most spatial database query languages were designed for expressing queries about spatial objects existing in databases [Egen89, Goh89, Rou88]. Few designers [Guti88, Main90] consider the construction of new spatial objects from existing objects. In the design of GeoSAL, however, object construction operators such as UNION, DIFFERENCE, and INTERSECTION play an important role. Their use in spatial query applications is illustrated in the examples below.

Let relations SR1 and SR2 represent field and soil layers, with the following definitions:

```
DEFINE TYPE FLayer := Tessellation [attr1:Literal];
DEFINE TYPE SLayer := Tessellation [attr1:Literal];
DEFINE TYPE FSLayer := Tessellation [attr1:Literal, attr2:Literal];
DEFINE OBJECT SR1:FLayer, SR2:SLayer;
```

Query 1: Find the fields where vegetables are grown. Assume for simplicity that potatoes and tomatoes are known to be the only vegetable varieties.

```
VSR:FLayer := SR1 WHERE (attr1 = 'potato') OR (attr1 = 'tomato')
[pg::UNION(pg), attr1:'vegetable'];
```

This query reclassifies fields of potatoes and tomatoes into fields of vegetable and, if fields meet, merges them into larger ones.

Query 2: Find fields of non-vegetable crops which do not grow in red soil.

```
LSR:FLayer := SR1 WHERE (attr1 <> 'potato') AND (attr1 <> 'tomato')
[pg::(pg DIFFERENCE (SR2 WHERE (attr1 = 'red') UNIO(pg)), attr1);
```

Query 3: Find fields of maize growing on red soil.

```
LSR1:FSLayer := *(a:SR1, b:SR2) WHERE (a.attr1 = 'maize') AND (b.attr1 = 'red')
[pg::(a.pg INTERSECTION b.pg), attr1:a.attr1, attr2:b.attr1];
```
5.2 A simplified site selection example

Site selection [Dang83, Smit83] is a typical spatial analysis task for which early GISs were designed. In principle, the analysis procedure can be described as follows. Given a set of input layers over the same area. Combine these layers location-by-location based on a set of previously defined selection criteria, and produce a suitability map containing candidate areas suitable for the siting. From the candidate areas, decision-makers choose the final site.

In the site selection task the operations reclassification, generalization and overlay play important roles in input layer preparation, combination and suitability map generation. In the following we use a simplified example to illustrate the use of these operations and to model the analysis procedure using GeoSAL queries.

5.2.1 The task

A county government intends to build a public golf course which needs more than 4 square kilometers of land. The selection criteria are:

(1) only land which is flat and owned by the state can be used,

(2) the best choice is land covered by grass or bare soil,

(3) agricultural land can not be used.

Three input layers are available: Landcover, Landowner and Landscape.

Landcover contains regions classified as forest, crops, grass and bare soil. Landowner is a partitioning of land based on ownership and Landscape is a classification of the landscape into flat and non-flat regions.

5.2.2 Analysis procedure with GeoSAL

Let the input layers be defined by:

```
DEFINE OBJECT Landcover: Tessellation [coverage:Literal],
  Landowner: Tessellation [owner:Literal],
  Landscape: Tessellation [type:Literal];
```

Query 1: Select from Landcover regions covered by forest, grass, and bare soils and reclassify grass and bare soils into open land.

```
Openland:Landcover := Landcover WHERE[coverage <> 'crops']
  [pg, coverage:IF coverage = 'grass' OR coverage = 'bare'
  THEN 'open' ELSE coverage];
```

Query 2: Merge adjacent polygons with the coverage value "open".

```
Openland1:Landcover := GENERALIZE(Openland);
```

Query 3: Overlay Landowner and Landscape to produce a set of polygons which are flat and owned by the state.

```
StateFlat:Tessellation[] := *(o:Landowner, s:Landscape)
  WHERE[(o.owner='state') AND (s.type='flat')]
  [pg:=(o.pg INTERSECTION s.pg)];
```
Query 4: Overlay Openland1 and StateFlat to produce a set of polygons with acceptable non-spatial properties.

\[
\text{Suitable1:Landcover} := \forall (o: \text{Openland1}, s: \text{StateFlat}) \\
\quad [\text{pg} := (o.\text{pg} \text{ INTERSECTION } s.\text{pg}), \text{coverage} := o.\text{coverage}]
\]

Query 5: Select open regions with an area greater than 4 square kilometers, i.e. 4 million square meters.

\[
\text{MostSuitable:Landcover} := \text{Suitable1 WHERE}[(\text{coverage} = \text{'open'}) \text{ AND (Area(pg) > 4E6})] \\
\quad [\text{pg, coverage}];
\]

If there is no area larger than 4 square kilometers, merge the open regions and the forest regions.

Query 6: Reclassify the category "forest" to "open".

\[
\text{Suitable2:Landcover} := \text{Suitable1 \{pg, coverage:'open'}];
\]

Query 7: Merge adjacent regions.

\[
\text{OpenAndForest:Landcover} := \text{GENERALIZE(Suitable2});
\]

Query 8: Select regions greater than 4 square kilometers.

\[
\text{Suitable:Tessellation}[] := \text{OpenAndForest WHERE}[(\text{Area(pg) > 4E6})] \\
\quad [\text{pg}];
\]

Relations MostSuitable and Suitable can be displayed graphically and an appropriate object chosen as the golf course.

6. Summary and conclusions

When properly adapted to spatial analysis applications, the query language approach can be expected to reduce considerably the number of concepts needed in future spatial analysis systems. This should amount to a significant increase in the expressive power and usefulness of these systems. The improvement will come as a consequence of the capabilities of abstraction, substitution, and combination of a systematic language approach.

In this paper, some examples have been given which, in the spirit of Tomlin [Toml90], illustrate how a spatial database language can be defined to fit the needs of spatial analysis applications.

It remains to be shown how to implement the language efficiently, as well as to test its ability to adapt to the requirements of realistic spatial analysis applications.

Also, one needs to explore the relative merits of and appropriate division of tasks between the "traditional" algebraic approach used in GeoSAL and the several visual language proposals that have been made recently [Goh89, Main90, Ange90].

ACKNOWLEDGEMENTS

The authors wish to thank the members of the joint FOA-NobelTech-ERA project "Decision support with GIS" for stimulating discussions and in particular Per Grape for reading the manuscript and providing valuable suggestions. We are also grateful to M. Mainguenaud for pointing out to us several ambiguities in the original manuscript.
REFERENCES


[Good90b] Goodchild, M. F., Tutorial on spatial data analysis at the 4th International Symposium on Spatial Data Handling, Zurich, 1990 (lecture notes).


