SPATIAL DATA TYPES

Markus Schneider
Department of Computer & Information Science & Engineering
University of Florida
Gainesville, FL 32611, USA
mschneid@cise.ufl.edu

SYNONYMS

Geometric Data Types

DEFINITION

Data types are a well known concept in computer science (for example, in programming languages or in database systems). A *data type* defines a set of homogeneous values and the allowable operations on those values. An example is a type *integer* representing the set of 32-bit integers and including operations such as addition, subtraction, and multiplication that can be performed on integers. *Spatial data types* or *geometric data types* provide a fundamental abstraction for modeling the geometric structure of objects in space as well as their relationships, properties, and operations. They are of particular interest in *spatial databases* [5, 8, 13] and *Geographical Information Systems* [14]. One speaks of *spatial objects* as values of spatial data types. Examples are two-dimensional data types for *points* (for example, representing the locations of lighthouses in the U.S.), *lines* (for example, describing the ramifications of the Nile Delta), *regions* (for example, depicting air-polluted zones), *spatial networks* (for example, representing the routes of the Metro in New York), and *spatial partitions* (for example, describing the 50 states of the U.S. and their exclusively given topological relationships of adjacency or disjointedness) as well as three-dimensional data types for *surfaces* (for example, modeling the shape of landscapes) or *volumes* (for example, representing urban areas). Operations on spatial data types include *spatial operations* like the geometric *intersection*, *union*, and *difference* of spatial objects, *numerical operations* like the *length* of a line or the *area* of a region, *topological relationships* checking the relative position of spatial objects to each other like *overlap*, *meet*, *disjoint*, or *inside*, and *cardinal direction relationships* like *north* or *southeast*.

HISTORICAL BACKGROUND

In the late seventies, the interest arose to store geometric data into databases. The success and efficiency of relational database technology for standard applications, which is rooted in its simple data model, its high-level query languages, and its well understood underlying theory, has led to many proposals to transfer this technology directly to geometric applications and to explicitly model the structure of spatial data as relations (tables). The consequence is that the user conceives spatial data in tabular form, just the same as standard data, and that a spatial object is represented by several or even many tuples. An example of such a relation schema is *RelName*(*id* : *integer*, x_1 : *integer*, y_1 : *integer*, x_2 : *integer*, y_2 : *integer*, *type* : *string*, <other information>) where x_1, y_1, x_2, and y_2 are the coordinates of a point or a line segment. The flag *type* indicates whether a tuple describes a point, a single line, a line segment of a line, or a line segment of a polygon. The value *id* denotes the object identifier.

This approach has revealed a number of fundamental drawbacks. Since all lines and polygons are decomposed into a set of line segments (tuples) scattered over a relation, a spatial object is not treated as an entity or unit but only *corresponds* to several tuples. This is different compared to values of standard data types. A second drawback is that the approach forces the user to model complex spatial objects in flat, independent relations. Since the representations of spatial data occurs on a very low level and is exclusively based on standard domains like integers, strings, and reals (while the user has originally intended to deal with points, lines, or polygons (regions)), an adequate treatment of spatial data is impeded. Although the facilities of the query language of a

DBMS are available, they are only of limited use. Since such a language is based on standard domains and has no concept of spatial data types, it cannot provide and support any meaningful geometric operations. A more detailed discussion can be found in [9].

SCIENTIFIC FUNDAMENTALS

The numerous deficiencies of the approach of modeling spatial data as relations have resulted in the assessment that this approach is unsuitable to manage spatial data in a clean and efficient manner and that a high-level view of spatial objects is essential. This has led to the design of spatial data types that are represented as *abstract data types*, thus provide such a high-level view, and can be used as attribute data types in a database schema in the same way as standard data types like *integer*, *float*, or *string*. That is, the internal structure of a spatial object is hidden from the user, and its features can only be retrieved by (abstract) operations on this object.

One can distinguish different kinds of spatial data types. *Universal spatial data types* either only provide a single generic spatial data type called *spatial* and therefore do not consider the dimensionality and shape of spatial objects, or they provide the types *spatial_0*, *spatial_1*, *spatial_2*, and *spatial_3* and thus consider the dimensionality but not the shape of spatial objects [2]. Another conceptual model for spatial data types is based on mathematical abstractions called *point sets*. The user is supplied with the concept that each spatial object consists of an infinite set of points that can be described by finite means. The approach in [7] introduces a type POINT-SET for point sets together with a collection of geometric operations. Aspects like dimensionality and shape of an object are not considered. A further approach of modeling spatial objects is that of using *half planes* [12] where each half plane is defined by a *half plane segment*. A half plane segment uniquely determines a straight line which is given by an inequality, passes this segment and forms the one-sided boundary of a half plane. For constructing a polygonal region, an appropriately arranged sequence of intersection operations (conjunction of inequalities) defined on half planes is employed. This concept is the precursor of so-called *constraint spatial databases*. Most popular and fundamental abstractions of spatial objects fall into the category of *structure-based spatial data types*. These data types organize space into points, lines, regions, surfaces, volumes, spatial partitions, spatial networks, and similarly structured entities. Thus, this approach considers the structural shape and spatial extent of spatial objects, that is, their geometry. Spatial data types for points, lines, and regions have, for example, been considered in [1, 4, 6, 9, 10, 15], for surfaces and volumes in [11], for spatial partitions in [3], and for spatial networks in [13].

Structure-based spatial data types have prevailed and form the basis of a large number of data models and query languages for spatial data. They also have found broad acceptance in spatial extension packages of commercially and publicly available database systems as well as in Geographical Information Systems. One can distinguish the older generation of *simple* spatial data types and the newer generation of *complex* spatial data types, depending on the spatial complexity the types are able to model. In the two-dimensional space, simple spatial data types only provide simple object structures like single points, continuous lines, and simple regions (Figure 1(a)-(c)). However, from an application perspective, simple spatial data types have turned out to be inadequate abstractions for spatial applications since they are insufficient to cope with the variety and complexity of geographic reality. From a formal perspective, they are not closed under the geometric set operations *intersection*, *union*, and *difference*. This means that these operations applied to two simple spatial objects can produce a spatial object that is *not* simple. Complex spatial data types solve these problems. They provide universal and versatile spatial objects and are closed under geometric set operations. They allow objects with multiple components, region components that may have holes, and line components that may model ramified, connected geometric networks (Figure 1(d)-(f)).

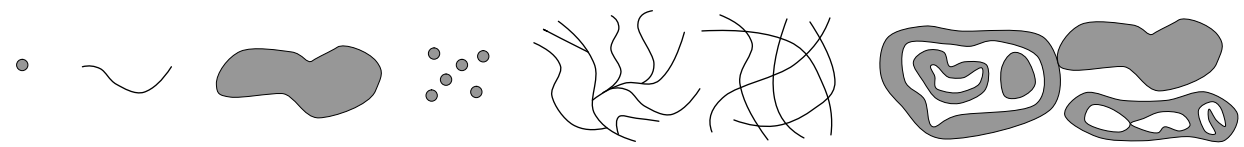
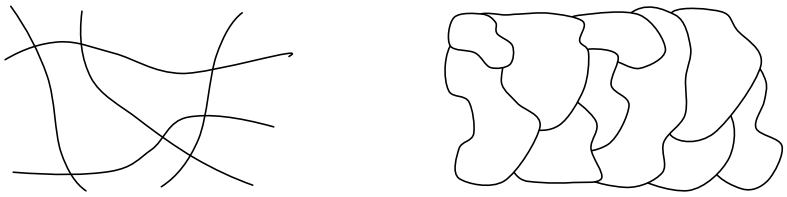


Figure 1: Examples of a simple point object (a), a simple line object (b), a simple region object (c), a complex point object (d), a complex line object (e), and a complex region object (f).



(a) (b)

Figure 2: Examples of a spatial network (a) and a spatial partition (b).

Even more complex structure-based spatial data types are spatial networks and spatial partitions. They are the essential components of maps. A *spatial network* (Figure 2(a)) can be viewed as a spatially embedded graph which consists of a set of point objects representing its nodes and a set of line objects describing the geometry of its edges. Examples are highways, rivers, public transport systems, power lines, and phone lines. A *spatial partition* (Figure 2(b)) is a set of region objects together with the topological constraint that any two regions either meet or are disjoint. The neighborhood relationship is of particular interest here since region objects may share common boundaries. Examples are states, school districts, crop fields, and land parcels. Both in spatial networks and in spatial partitions, their components (line objects, region objects) are annotated with thematic data like state name, unemployment rate, and parcel id.

Spatial operations[1] manipulate spatial objects. They take spatial objects as operands and return either spatial objects or scalar values (like Boolean or numerical values) as results. One can classify them into the following categories:

Spatial predicates returning Boolean values. A spatial relationship is a relationship between two or more spatial objects. A *spatial predicate* compares two spatial objects with respect to some spatial relationship and thus conforms to a binary relationship returning a Boolean value. Spatial predicates can be classified into three subcategories. *Topological predicates* characterize the relative position of spatial objects towards each other and are preserved under topological transformations such as translation, rotation, and scaling; they do not depend on metric concepts like distance. Examples are the well known predicates *equal*, *disjoint*, *coveredBy*, *covers*, *overlap*, *meet*, *inside*, and *contains* between two simple regions. *Metric predicates* use measurements such as distances. For example, the predicates *in_circle* and *in_window* test if a spatial object is located within the scope of a predefined circle or rectangle. *Directional predicates* like *north* or *southeast* compare the cardinal direction of a target object with respect to a reference object.

Spatial operations returning numbers. These operations compute metric properties of spatial objects and return a number. Examples are the operations *area* and *perimeter* computing the corresponding values of a region object, the operation *length* calculating the total length of a line object, the operation *diameter* determining the largest distance between any two object components, the operation *dist* computing the minimal distance between two spatial objects, and the operation *cardinality* yielding the number of components of a spatial object.

Spatial operations returning spatial objects. These operations return spatial objects as results and can be subdivided into object construction operations, which construct new objects from existing objects, and object transformation operations, which transform one or more spatial objects into a new spatial object. The *object construction operations* include, for example, the *geometric set operations union*, *intersection*, and *difference*, which satisfy closure properties, the operation *convex_hull*, which constructs the smallest convex region (polygon) enclosing a finite collection of points, the operation *boundary*, which returns the boundary of a region object as a line object, the operation *box*, which determines the minimal, axis-parallel rectangle (called *minimal bounding box* or *rectangle*) that bounds a spatial object, and the operation *components*, which extracts the vertices of a line object. Examples of *object transformation operations* are the operation *extend*, which takes a spatial object s and a real number r as operands and creates a polygonal region that is a spatial extension of s with distance r from s (also known as *buffer zoning*), the operation *rotate*, which rotates a spatial object around a point, and the operation *translate*, which moves a spatial object by a defined vector.

[1]The reader is referred to the Section CROSS REFERENCES that lists a number of entries describing spatial operations in great detail. This section only gives an overview and a brief characterization of spatial operations.

Spatial operations on spatial networks and spatial partitions. An important operation on spatial networks is the *shortest_path* operator. It computes the route or path of minimum distance between a source and a destination. An important operation on spatial partitions is the *overlay* operation. It takes two spatial partitions modeling different themes as operands, lays them transparently on top of each other, and combines them into a new spatial partition by intersection. A large collection of other operations is available for both kinds of structures.

A brief example illustrates the embedding of a spatial data type into a relation schema and the posing of a spatial query. Consider the map of the 50 states of the USA. Besides its thematic attributes like name and population, each state is also described by a geometry which is a region. Cities can be represented as points, that is, one is here interested in their location and not so much in their extent. As thematic attributes, one could be interested in their name and population. In the following two relation schemas, the spatial data types *point* and *region* are used in the same way as attribute data types as standard data types.

```
states(sname: string, spop: integer, territory: region)
cities(cname: string, cpop: integer, loc: point)
```

A query could ask for all pairs of city names and state names where a city is located in a state. This can then be formulated as a *spatial join*:

```
select cname, sname
from cities, states
where loc inside territory
```

The term *inside* is a topological predicate testing whether a point object is located inside a region object.

KEY APPLICATIONS

Spatial data types are a universal and general concept for representing geometric information in all kinds of spatial applications. Hence, they are not only applicable to a few key applications. In principle, all applications in the geosciences (for example, geography, hydrology, soil sciences) and Geographical Information Systems as well as many applications in government and administration (for example, cadastral application, urban planning) can benefit from them. Independent studies have shown that about 80% of all data have spatial features (like geometric attributes) or a spatial reference (like an address). Thus, it is not surprising that independent international studies have predicted that geoinformation technology will belong to the most important and promising technologies in the future, besides biotechnology and nanotechnology.

CROSS REFERENCES

SPATIAL OPERATIONS AND MAP OPERATIONS
SIMPLICIAL COMPLEXES
TOPOLOGICAL RELATIONSHIPS
DIMENSION-EXTENDED TOPOLOGICAL RELATIONSHIPS
CARDINAL DIRECTION RELATIONSHIPS
THREE-DIMENSIONAL GIS

RECOMMENDED READING

[1] E. Clementini and P. Di Felice. A Model for Representing Topological Relationships between Complex Geometric Features in Spatial Databases. *Information Systems*, 90(1-4):121–136, 1996.

[2] M. J. Egenhofer. Spatial SQL: A Query and Presentation Language. *IEEE Trans. on Knowledge and Data Engineering*, 6(1):86–94, 1994.

[3] M. Erwig and M. Schneider. Partition and Conquer. In *3rd Int. Conf. on Spatial Information Theory*, LNCS 1329, pages 389–408. Springer-Verlag, 1997.

[4] R. H. Güting. Geo-Relational Algebra: A Model and Query Language for Geometric Database Systems. In *Int. Conf. on Extending Database Technology*, pages 506–527, 1988.

[5] R. H. Güting. An Introduction to Spatial Database Systems. *VLDB Journal*, 3(4):357–399, 1994.

[6] R. H. Güting and M. Schneider. Realm-Based Spatial Data Types: The ROSE Algebra. *VLDB Journal*, 4:100–143, 1995.

[7] F. Manola and J. A. Orenstein. Toward a General Spatial Data Model for an Object-Oriented DBMS. In *12th Int. Conf. on Very Large Data Bases*, pages 328–335, 1986.

[8] P. Rigaux, M. Scholl, and Agnes Voisard. *Spatial Databases – With Applications to GIS*. Morgan Kaufmann Publishers, 2002.

[9] M. Schneider. *Spatial Data Types for Database Systems - Finite Resolution Geometry for Geographic Information Systems*, volume LNCS 1288. Springer-Verlag, 1997.

[10] M. Schneider and T. Behr. Topological Relationships between Complex Spatial Objects. *ACM Trans. on Database Systems*, 31(1):39–81, 2006.

[11] M. Schneider and B. Weinrich. An Abstract Model of Three-Dimensional Spatial Data Types. In *12th ACM Symp. on Geographic Information Systems*, pages 67–72, 2004.

[12] M. Scholl and A. Voisard. Thematic Map Modeling. In *1st Int. Symp. on Advances in Spatial Databases*, pages 167–190, 1989.

[13] S. Shekar and S. Chawla. *Spatial Databases: A Tour*. Prentice Hall, 2003.

[14] M. F. Worboys and M. Duckham. *GIS: A Computing Perspective*. CRC, 2004.

[15] M.F. Worboys and P. Bofakos. A Canonical Model for a Class of Areal Spatial Objects. In *3rd Int. Symp. on Advances in Spatial Databases*, LNCS 692, pages 36–52. Springer-Verlag, 1993.