

# An Introduction to Spatial Databases

R. H. Guting

VLDB Journal v3, n4, October 1994

Speaker: Giovanni Conforti

## Outline:

a rather old (but quite complete) survey on Spatial DBMS

- Introduction & definition
- Modeling
- Querying
- Data structures & algorithms
- System architecture

# Introduction

- A common technology for some Applications:
  - GIS (geographic/geo-referenced data)
  - VLSI design (geometric data)
  - modeling complex phenomena (spatial data)
- All need to manage large collections of relatively simple spatial objects
- Spatial DB vs. Image/pictorial DB [1990]
  - Spatial DB contains objects “ in ” the space
  - Image DB contains representations “ of ” a space (images, pictures,... : raster data)

# SDBMS Definition

A spatial database system:

- Is a database system
  - A DBMS with additional capabilities for handling spatial data
- Offers spatial data types (SDTs) in its data model and query language
  - Structure in space: e.g., POINT, LINE, REGION
  - Relationships among them: (*l intersects r*)
- Supports SDT in its implementation providing at least
  - spatial indexing (retrieving objects in particular area without scanning the whole space)
  - efficient algorithms for spatial joins (not simply filtering the cartesian product)

# Modeling

Assume 2-D and GIS application, two basic things need to be represented:

- Objects in space: cities, forests, or rivers  
    ▶ *single objects*
- *Coverage/Field*: say something about every point in space (e.g., partitions, thematic maps)  
    ▶ *spatially related collections of objects*

## Modeling: spatial primitives for objects

- Point: object represented only by its location in space, e.g. center of a state
- Line (actually a curve or ployline): representation of moving through or connections in space, e.g. road, river
- Region: representation of an extent in 2d-space, e.g. lake, city

## Modeling: coverages

- Partition: set of **region** objects that are required to be disjoint (adjacency or region objects with common boundaries), e.g. thematic maps
- Networks: embedded graph in plane consisting of set of points (vertices) and lines (edges) objects, e.g. highways, power supply lines, rivers

## Modeling: a sample spatial type system (1/2)

EXT={lines, regions}, GEO={points, lines, regions}

- Spatial predicates for topological relationships:
  - **inside**:  $geo \times regions \rightarrow bool$
  - **intersect, meets**:  $ext1 \times ext2 \rightarrow bool$
  - **adjacent, encloses**:  $regions \times regions \rightarrow bool$
- Operations returning atomic spatial data types:
  - **intersection**:  $lines \times lines \rightarrow points$
  - **intersection**:  $regions \times regions \rightarrow regions$
  - **plus, minus**:  $geo \times geo \rightarrow geo$
  - **contour**:  $regions \rightarrow lines$

## Modeling: a sample spatial type system (2/2)

- Spatial operators returning numbers
  - **dist**:  $geo1 \times geo2 \rightarrow real$
  - **perimeter, area**:  $regions \rightarrow real$
- Spatial operations on set of objects
  - **sum**:  $set(obj) \times (objgeo) \rightarrow geo$
  - A spatial aggregate function, geometric union of all attribute values, e.g. union of set of provinces determine the area of the country
  - **closest**:  $set(obj) \times (objgeo1) \times geo2 \rightarrow set(obj)$
  - Determines within a set of objects those whose spatial attribute value has minimal distance from geometric query object
  - Other complex operations: overlay, buffering, ...

## Modeling: spatial relationships

- *Topological* relationships: e.g. adjacent, inside, disjoint. Are invariant under topological transformations like translation, scaling, rotation
- *Direction* relationships: e.g. above, below, or north\_of, sothwest\_of, ...
- *Metric* relationships: e.g. distance

6 valid topological relationships between two simple regions (no holes, connected): disjoint, in, touch, equal, cover, overlap

## Modeling: SDBMS data model

- DBMS data model must be extended by SDTs at the level of atomic data types (such as integer, string), or better be open for user-defined types (OR-DBMS approach):

**relation** states (sname: STRING; area: REGION; spop: INTEGER)

**relation** cities (cname: STRING; center: POINT; ext: REGION; cpop: INTEGER);

**relation** rivers (rname: STRING; route: LINE)

## Querying

- Two main issues:
  1. Connecting the operations of a spatial algebra (including predicates for spatial relationships) to the facilities of a DBMS query language. Fundamental spatial algebra operator are:
    - Spatial selection
    - Spatial join
    - ... (overlay, fusion)
  2. Providing graphical presentation of spatial data (i.e. results of queries), and graphical input of SDT values used in queries.

## Querying: spatial selection

- *Spatial selection*: returning those objects satisfying a spatial predicate with the query object
  - “All cities in Bavaria”
 

```
SELECT sname FROM cities c WHERE c.center inside Bavaria.area
```
  - “All rivers intersecting a query window”
 

```
SELECT * FROM rivers r WHERE r.route intersects Window
```
  - “All big cities no more than 100 Kms from Hagen”
 

```
SELECT cname FROM cities c
WHERE dist(c.center, Hagen.center) < 100 and c.pop > 500k
(conjunction with other predicates and query optimization)
```

## Querying: spatial join

- *Spatial join*: A join which compares any two joined objects based on a predicate on their spatial attribute values.
  - “For each river pass through Bavaria, find all cities within less than 50 Kms.”
 

```
SELECT r.rname, c.cname, length(intersection(r.route, c.area))
FROM rivers r, cities c
WHERE r.route intersects Bavaria.area and dist(r.route,c.area) < 50
```

## Querying: I/O (1/2)

- Graphical I/O issue: how to determine “Window” or “Bavaria” in previous examples (input); or how to show “intersection(route, Bavaria.area)” or “r.route” (output) (results are usually a combination of several queries).
- Requirements for spatial querying [Egenhofer]:
  - Spatial data types
  - Graphical display of query results
  - Graphical combination (overlay) of several query results (start a new picture, add/remove layers, change order of layers)
  - Display of context (e.g., show background such as a raster image (satellite image) or boundary of states)
  - Facility to check the content of a display (which query contributed to the content)

## Querying: I/O (2/2)

### Other requirements for spatial querying [Egenhofer]:

- Extended dialog: use pointing device to select objects within a subarea, zooming, ...
- Varying graphical representations: different colors, patterns, intensity, symbols to different objects classes or even objects within a class
- Legend: clarify the assignment of graphical representations to object classes
- Label placement: selecting object attributes (e.g., population) as labels
- Scale selection: determines not only size of the graphical representations but also what kind of symbol be used and whether an object be shown at all
- Subarea for queries: focus attention for follow-up queries

## Data Structures & Algorithms

1. Implementation of spatial algebra in an integrated manner with the DBMS query processing.
2. Not just simply implementing atomic operations using computational geometry algorithms, but consider the use of the predicates within set-oriented query processing **Spatial indexing** or access methods, and **spatial join** algorithms

## Data Structures (1/3)

- Representation of a value of a SDT must be compatible with two different views:
  1. DBMS perspective:
    - Same as attribute values of other types with respect to generic operations
    - Can have varying and possibly large size
    - Reside permanently on disk page(s)
    - Can efficiently be loaded into memory
    - Offers a number of type-specific implementations for generic operations needed by the DBMS (e.g., transformation functions from/to ASCII or graphic)

## Data Structures (2/3)

### 2. Spatial algebra implementation perspective, the representation:

- Is a value of some programming language data type
- Is some arbitrary data structure which is possibly quite complex
- Supports efficient computational geometry algorithms for spatial algebra operations
- Is not geared only to one particular algorithm but is balanced to support many operations well enough

## Data Structures (3/3)

- From both perspectives, the representation should be mapped by the compiler into a single or perhaps a few contiguous areas (to support DBMS paging). Also supports:
  - Plane sweep sequence: object's vertices stored in a specific sweep order (e.g. x-order) to expedite plane-sweep operation.
  - Approximations: stores some approximations as well (e.g. MBR) to speed up operations (e.g. comparison)
  - Stored unary function values: such as perimeter or area be stored once the object is constructed to eliminate future expensive computations.

# Spatial Indexing

- To expedite spatial selection (as well as other operations such as spatial joins, ...)
- It organizes space and the objects in it in some way so that only parts of the objects need to be considered to answer a query.
- Two main approaches:
  1. Dedicated spatial data structures (e.g. R-tree)
  2. Spatial objects mapped to a 1-D space to utilize standard indexing techniques (e.g. B-tree)

## Spatial Indexing: operations

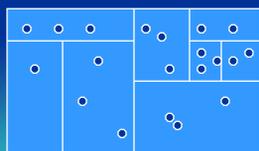
- Spatial data structures either store *points* or *rectangles* (for line or region values)
- Operations on those structures: insert, delete, member
- Query types for points:
  - Range query: all points within a query rectangle
  - Nearest neighbor: point closest to a query point
  - Distance scan: enumerate points in increasing distance from a query point.
- Query types for rectangles:
  - Intersection query
  - Containment query

## Spatial Indexing: idea... approximate!

- A fundamental idea: use of approximations as keys
  - 1) continuous (e.g. bounding box)
  - 2) Grid (a geometric entity as a set of cells).
- Filter and refine strategy for query processing:
  1. Filter: returns a set of candidate object which is a superset of the objects fulfilling a predicate
  2. Refine: for each candidate, the exact geometry is checked

## Spatial Indexing: memory organization

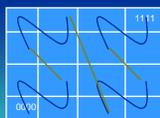
- A spatial index structure organizes points into buckets.
- Each bucket has an associated *bucket region*, a part of space containing all objects stored in that bucket.
- For point data structures, the regions are disjoint & partition space so that each point belongs into precisely one bucket.
- For rectangle data structures, bucket regions may overlap.



A kd-tree partitioning of 2d-space where each bucket can hold up to 3 points

## Spatial Indexing: 1-D Grid approx. (1/2)

- One dimensional embedding: z-order or bit-interleaving
  - Find a linear order for the cells of the grid while maintaining “locality” (i.e., cells close to each other in space are also close to each other in the linear order)
  - Define this order recursively for a grid that is obtained by hierarchical subdivision of space



## Spatial Indexing: 1-D Grid approx. (2/2)

- Any shape (approximated as set of cells) over the grid can now be decomposed into a *minimal* number of cells at different levels (using always the highest possible level)
- Hence, for each spatial object, we can obtain a set of “spatial keys”
- Index: can be a B-tree of lexicographically ordered list of the union of these spatial keys

## Spatial indexing: 2-D points

- Data structures representing points have a much longer tradition:
  - Kd-tree and its extensions (KDBtree and LSDtree)
  - Grid file (organizing buckets into an irregular grid of pointers)

## Spatial Indexing: 2-D rectangles

- Spatial index structures for rectangles: unlike points, rectangles don't fall into a unique cell of a partition and might intersect partition boundaries
  - Transformation approach: instead of k-dimensional rectangles, 2k-dimensional points are stored using a point data structure
  - Overlapping regions: partitioning space is abandoned & bucket regions may overlap (e.g. R-tree & R\*-tree)
  - Clipping: keep partitioning, a rectangle that intersects partition boundaries is clipped and represented within each intersecting cell (e.g. R+-tree)

# Spatial Join

- Traditional join methods such as hash join or sort/merge join are not applicable.
- Filtering cartesian product is expensive.
- Two general classes:
  1. Grid approximation/bounding box
  2. None/one/both operands are presented in a spatial index structure
- Grid approximations and overlap predicate:
  - A parallel scan of two sets of z-elements corresponding to two sets of spatial objects is performed
  - Too fine a grid, too many z-elements per object (inefficient)
  - Too coarse a grid, too many “false hits” in a spatial join

# Spatial Join

- Bounding boxes: for two sets of rectangles  $R$ ,  $S$  all pairs  $(r,s)$ ,  $r \in R$ ,  $s \in S$ , such that  $r$  intersects  $s$ :
  - **No spatial index on  $R$  and  $S$** : *bb\_join* which uses a computational geometry algorithm to detect rectangle intersection, similar to external merge sorting
  - **Spatial index on either  $R$  or  $S$** : *index join* scan the non-indexed operand and for each object, the bounding box of its SDT attribute is used as a search argument on the indexed operand (only efficient if non-indexed operand is not too big or else *bb\_join* might be better)
  - **Both  $R$  and  $S$  are indexed**: synchronized traversal of both structures so that pairs of cells of their respective partitions covering the same part of space are encountered together.

# System Architecture

- Extensions required to a standard DBMS architecture:
  - Representations for the data types of a spatial algebra
  - Procedures for the atomic operations (e.g. overlap)
  - Spatial index structures
  - Access operations for spatial indices (e.g. insert)
  - Filter and refine techniques
  - Spatial join algorithms
  - Cost functions for all these operations (for query optimizer)
  - Statistics for estimating selectivity of spatial selection and join
  - Extensions of optimizer to map queries into the specialized query processing method
  - Spatial data types & operations within data definition and query language
  - User interface extensions to handle graphical representation and input of SDT values

# System Architecture

- The only clean way to accommodate these extensions is an integrated architecture based on the use of an extensible DBMS.
- There is no difference in principle between:
  - a standard data type such as a STRING and a spatial data type such as REGION
  - same for operations: concatenating two strings or forming intersection of two regions
  - clustering and secondary index for standard attribute (e.g. B-tree) & for spatial attribute (R-tree)
  - sort/merge join and bounding-box join
  - query optimization (only reflected in the cost functions)

# An Introduction to Spatial Databases

R. H. Guting

VLDB Journal v3, n4, October 1994

Speaker: Giovanni Conforti

