Introduction to Spatial Database Systems

by Cyrus Shahabi

from

Ralf Hart Hartmut Guting's

VLDB Journal v3, n4, October 1994

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 …

- 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 fo generic operations needed by the DBMS (e.g., transformation functions from/to ASCII or graphic)

Data Structures …

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 no geared only to one particular algorithm but is balanced to support many operations well enough

Data Structures …

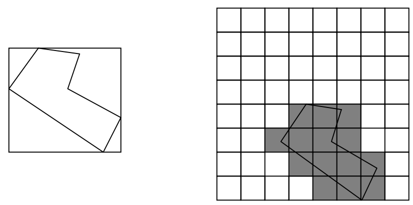
- From both perspectives, the representation should be mapped by the compiler into a single or perhaps a few contiguous areas (to support DBMS paging). Also supports:
- Plane sweep sequence: object's vertices stored in a specific sweep order (e.g., x-order) to expedite plane-sweep operation.
- Approximations: stores some approximations as well, e.g., MBR
- 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 space and a subset 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

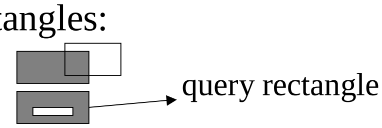
- A fundamental idea: use of approximations: 1) continuous (e.g., bounding box), or 2) grid.



- 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

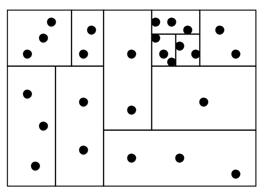
Spatial Indexing …

- 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 …

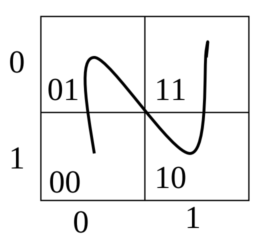
- A spatial index structure organizes points into buckets.
- Each bucket has an associated *bucket region*, a part of space containing all objects stored in that bucket.
- For point data structures, the regions are disjoint & partition space so that each point belongs into precisely one bucket.
- For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

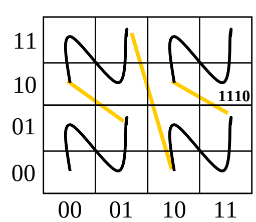


A kd-tree partitioning of 2d-space where each bucket can hold up to 3 points

Spatial Indexing …

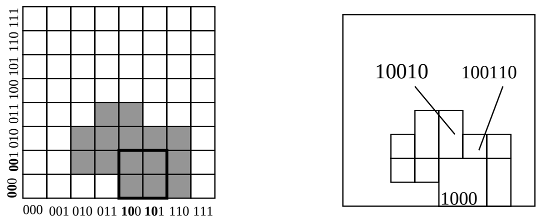
- One dimensional embedding: z-order or bit-interleaving
  - Find a linear order for the cells of the grid while maintaining “locality” (i.e., cells close to each other in space are also close to each other in the linear order)
  - Define this order recursively for a grid that is obtained by hierarchical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





Spatial Indexing …

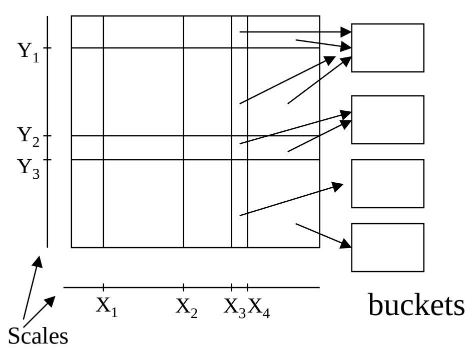
- Any shape (approximated as set of cells) over the grid can now be decomposed into a *minimal* number of cells at different levels (using always the highest possible level)



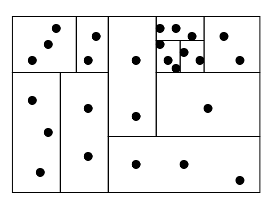
- Hence, for each spatial object, we can obtain a set of "spatial keys"
- Index: can be a B-tree of lexicographically ordered list of the union of these spatial keys

Spatial Indexing …

- Spatial index structures for points:



Grid-file



KD-Tree

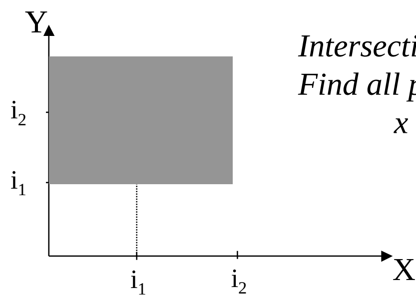
Spatial Indexing …

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 Indexing …

- A rectangle with 4 coordinates $(X_{left}, X_{right}, Y_{bottom}, Y_{top})$ can be considered as a point in 4d-space
- For illustration, consider how an interval $i = (i_1, i_2)$ with 2 coordinates can be mapped to 2d-space (as a point):



Intersection query with interval i:
Find all points (x,y) where:
$x < i_2$ *and* $y > i_1$

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)

System Architecture

Extensibility of the architecture is orthogonal to the data model implemented by that architecture:

- Probe is OO
- DASDBS is nested relational
- POSTGRES, Starbust and Gral extended relational models

- OO is good due to extensibility at the data type level, but lack extensibility at index structures, query processing or query optimization.
- Hence, current commercial solutions are OR-DBMSs:
  - NCR Teradata Object Relational (TOR)
  - IBM DB2 (spatial extenders)
  - Informix Universal Server (spatial datablade)
  - Oracle 8i (spatial cartridges)