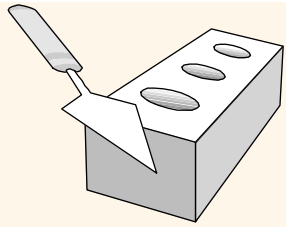
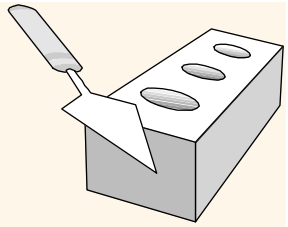


Spatial Database Systems

Excerpt from

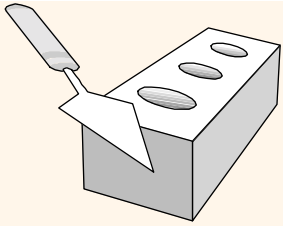
VLDB Journal Vol. 3, No. 4, October 1994 by Ralf Hartmut Guting

Outline

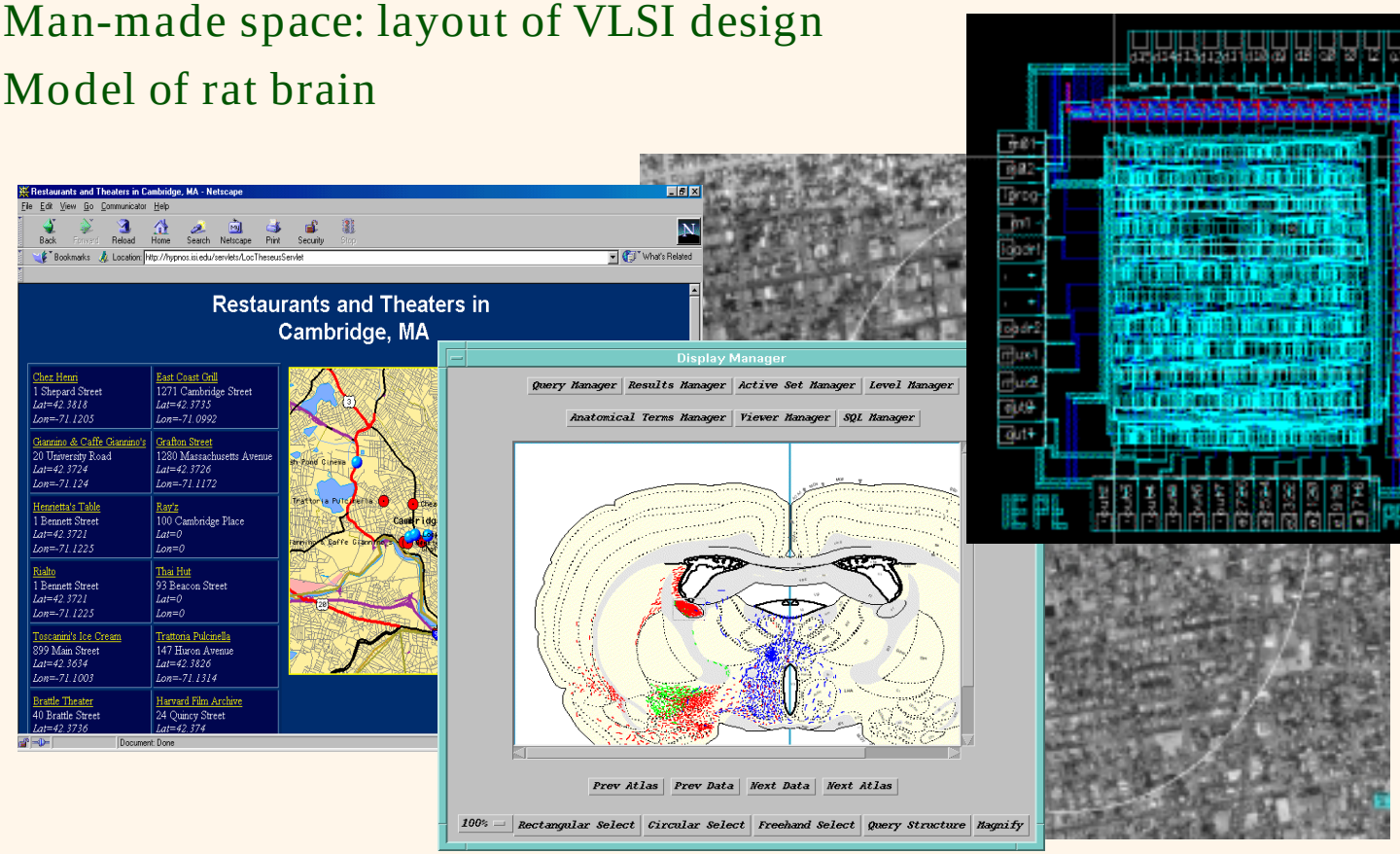


- Introduction
- Modeling
- Querying
- Data structures and algorithms
- System architecture

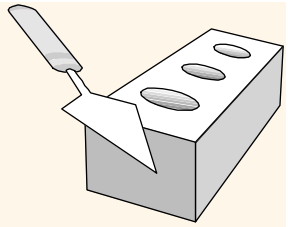
Spatial Database Applications



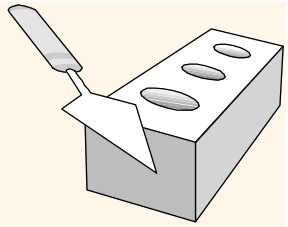
- Various fields / applications require management of geometric, geographic or *spatial* data:
  - A geographic space: surface of the earth
  - Man-made space: layout of VLSI design
  - Model of rat brain



What is NOT a Spatial Database!

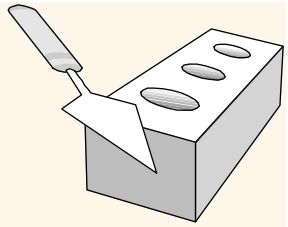


- Common challenge: dealing with large collections of relatively simple geometric objects
- Different from *image* and *pictorial* database systems:
  - Containing sets of objects in space rather than images or pictures of a space



What is a Spatial Database: Definition

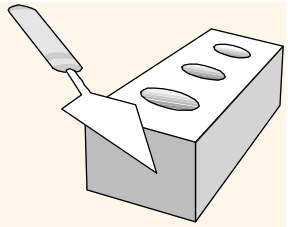
- A spatial database system:
  - Is a database system
    - A DBMS with additional capabilities for handling spatial data
  - Offers spatial data types (SDTs) in its data model and query language
    - Structures in space: e.g., POINT, LINE, REGION
    - Relationships among them: (*l intersects r*)
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



Modeling

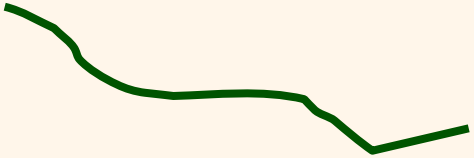
- Assuming 2D and GIS applications, two basic entities need to be modeled / represented:
  - Objects in space: e.g., cities, rivers, cars, …
    - ➔ Modeling *single objects*
  - Space: say something about every point in space (e.g., thematic maps that partition a country into districts)
    - ➔Modeling *spatially related collections of objects*

Modeling Single Objects

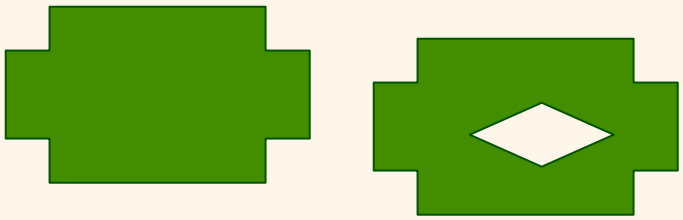


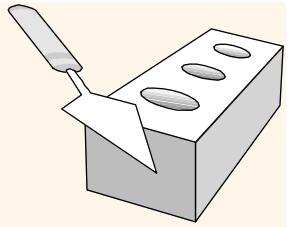
- Fundamental abstractions for modeling single objects:
  - Point: object represented only by its location in space, e.g., the capital of a state

  - Line (actually a curve or polyline): representation of a moving through or connections in space, e.g., road, river



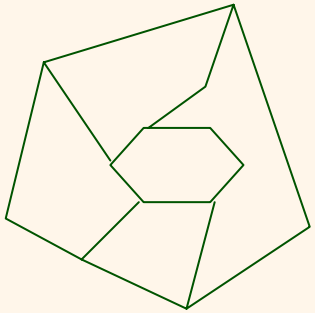
  - Region: representation of an extent in 2D space, e.g., lake, city



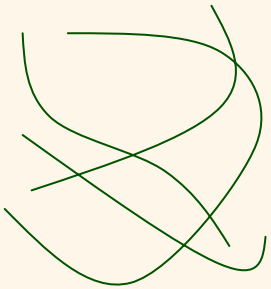


Modeling related Collections of Objects

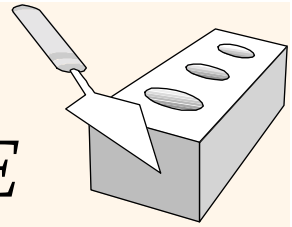
- Instances of spatially related collections of objects:
  - Partition: set of **region** objects that are adjacent and disjoint, e.g., thematic maps



  - Spatially Embedded Networks: embedded graph in plane consisting of a set of point (vertices) and line (edges) objects, e.g. highways, power lines, rivers, etc.



An Example Algebra to Model SDBs: ROSE

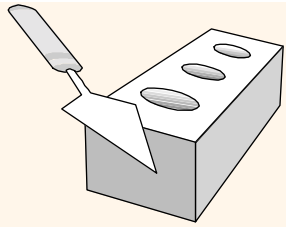


ROSE Algebra:

SDTs: Points, Lines, Regions; EXT={lines, regions}, GEO={points, lines, regions}

- Operators / Predicates that evaluate topological relationships:
  - **inside:** *geo* × *regions* → *bool*
  - **intersect:** *ext1* × *ext2* → *bool*
  - **adjacent:** *regions* × *regions* → *bool*
- Operators returning atomic spatial data types:
  - **intersection:** *lines* × *lines* → *points*
  - **intersection:** *regions* × *regions* → *regions*
  - **plus, minus:** *geo* × *geo* → *geo*
  - **contour:** *regions* → *lines*

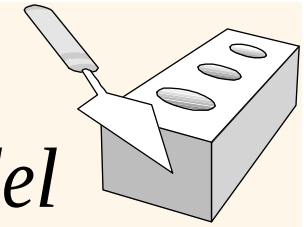
ROSE Algebra (cont'd)



- Operators returning numbers:
  - **dist:** *geo1* × *geo2* → *real*
  - **perimeter, area:** *regions* → *real*

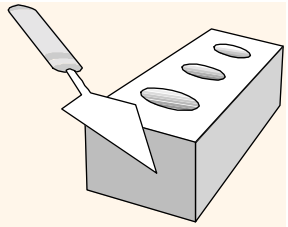
- Operators on set of objects (i.e., aggregate operators):
  - **sum:** *set(obj)* × *(obj* → *geo)* → *geo*
    - A spatial aggregate function, geometric union of all attribute values, e.g., union of set of provinces determine the area of the country
  - **closest:** *set(obj)* × *(obj* → *geo1)* × *geo2* → *set(obj)*
    - Determines within a set of objects those whose spatial attribute value has minimal distance from geometric query object

Spatial Relationships to Capture with a Model



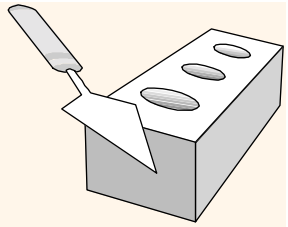
- Spatial relationships:
  - Topological relationships: e.g., *adjacent*, *inside*, *disjoint*; these are invariant under topological transformations such as translation, scaling, and rotation
  - Direction relationships: e.g., *above*, *below*, *or north_of*, *southwest_of*
  - Metric relationships: e.g., *distance*

Spatial Relationships (cont'd)



- Enumerating all possible topological relationships between two simple regions (no holes, connected) based on intersections of their boundaries (δA) and interiors (A^o):
  - 8 invalid cases,
  - 2 symmetric cases
  - 6 valid cases: *disjoint*, *in*, *touch*, *equal*, *cover*, *overlap*

$\partial A_1 \cap \partial A_2$	$\partial A_1 \cap A_2^\circ$	$A_1^\circ \cap \partial A_2$	$A_1^\circ \cap A_2^\circ$	relationship name
$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	A_1 disjoint A_2
$\emptyset$	$\emptyset$	$\emptyset$	$\neq \emptyset$	
$\emptyset$	$\emptyset$	$\neq \emptyset$	$\emptyset$	
$\emptyset$	$\emptyset$	$\neq \emptyset$	$\neq \emptyset$	A_2 in A_1
$\emptyset$	$\neq \emptyset$	$\emptyset$	$\emptyset$	
$\emptyset$	$\neq \emptyset$	$\emptyset$	$\neq \emptyset$	A_1 in A_2
$\emptyset$	$\neq \emptyset$	$\neq \emptyset$	$\emptyset$	
$\emptyset$	$\neq \emptyset$	$\neq \emptyset$	$\neq \emptyset$	
$\neq \emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	A_1 touch A_2
$\neq \emptyset$	$\emptyset$	$\emptyset$	$\neq \emptyset$	A_1 equal A_2
$\neq \emptyset$	$\emptyset$	$\neq \emptyset$	$\emptyset$	
$\neq \emptyset$	$\emptyset$	$\neq \emptyset$	$\neq \emptyset$	A_1 cover A_2
$\neq \emptyset$	$\neq \emptyset$	$\emptyset$	$\emptyset$	
$\neq \emptyset$	$\neq \emptyset$	$\emptyset$	$\neq \emptyset$	A_2 cover A_1
$\neq \emptyset$	$\neq \emptyset$	$\neq \emptyset$	$\emptyset$	
$\neq \emptyset$	$\neq \emptyset$	$\neq \emptyset$	$\neq \emptyset$	A_1 overlap A_2

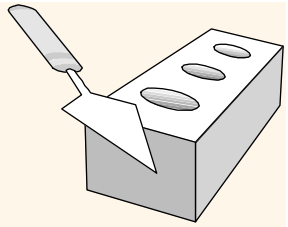


Querying

- Two main issues:

1. Connecting the operations of a spatial algebra (including predicates to express spatial relationships) to the facilities of a DBMS query language.
2. Providing graphical presentation of spatial data (i.e., results of queries) as well as graphical input of the SDT values used in queries.

Spatial Selection



Fundamental spatial algebra operations:

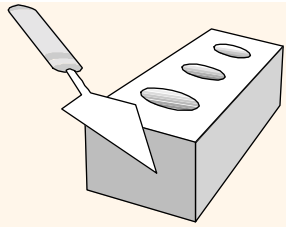
- ***Spatial selection***: returning those objects satisfying a spatial predicate with the query object.
  - **"All cities in Bavaria"**

    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

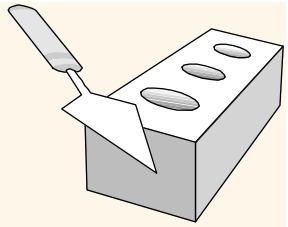
Spatial Join



- ***Spatial join***: A join which compares any two joined objects based on a predicate on their spatial attribute values.
  - **"For each river passing 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 Km
```

Graphical Presentation

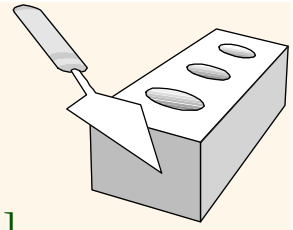


Graphical I/O issue:

- INPUT Presentation: e.g., how to define “Window” or “Bavaria”
- OUTPUT Presentation: how to show “intersection(route, Bavaria.area)” or “r.route”

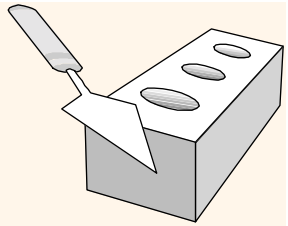
Requirements for Graphical Presentation

[Egenhofer, TKDE 94]



- Spatial data types
- Graphical display of query results
- Graphical combination (overlay) of several query results

  (start a new query, add / remove layers, change order of layers)
- Display of context (e.g., show background such as a raster image or boundary of states)
- Facility to check the content of a display (which query contributed to which overlay)
- 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



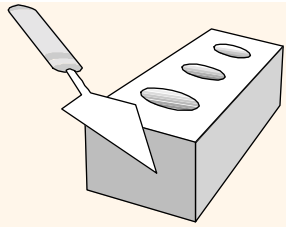
Data Structures & Algorithms

Need new data structures and algorithms for:

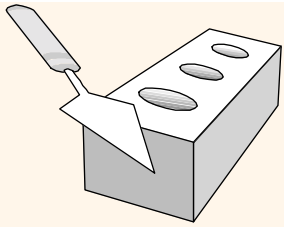
1. Implementation of spatial algebra in an integrated manner with the DBMS query processing.

2. Not just simply implementing atomic operations using computational geometry algorithms, but consider the use of the predicates within set-oriented query processing; thus, we need:
   - ➔ **Spatial indexes** or access methods, and
   - ➔ **Spatial join** algorithms

Spatial Indexing

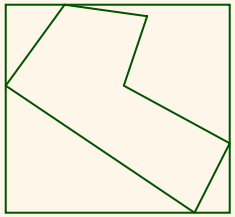


- To expedite spatial selection (as well as other operations such as spatial joins, …)
- It organizes space and the objects in it in some way so that only parts of the space and a subset of the objects need to be considered to answer a query.
- Two main approaches:
  1. Dedicated spatial data structures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

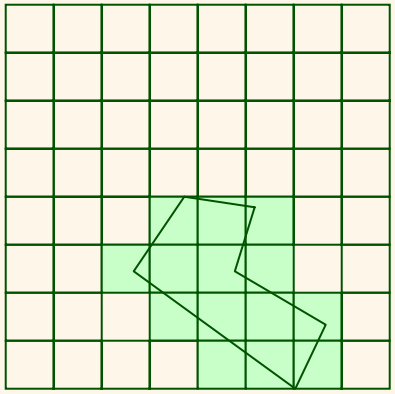


Approximation of Spatial Objects

- Approaches:
  - Continuous (e.g., bounding box)

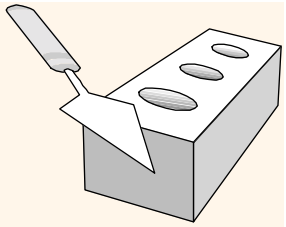


  - Grid

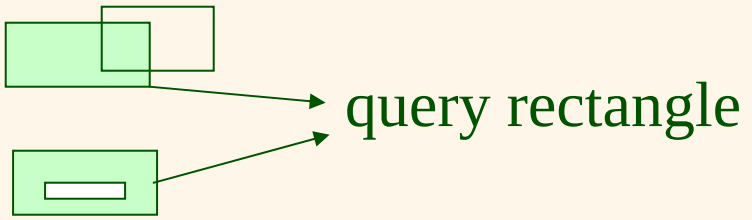


- Filter and refine strategy for query processing:
  1. Filter: returns a set of candidate object which is a superset of the objects satisfying a predicate;
  2. Refine: for each candidate, the exact geometry is checked

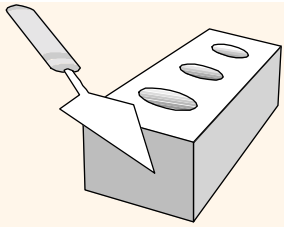
Interface of Spatial Indexes



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

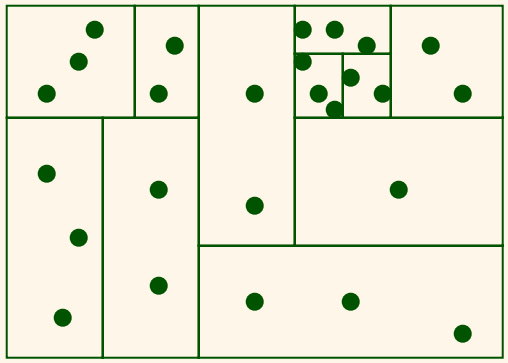


Indexing and Storage



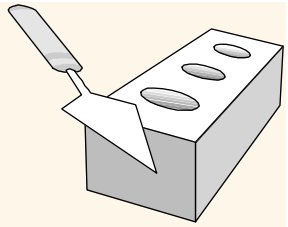
- A spatial index structure organizes points into *bucket regions*.
  - Each *bucket* maps to a disk block
  - Each bucket has an associated bucket region, a part of space containing all objects stored in that bucket
- For point data structures, the regions are disjoint & partition space so that each point belongs into precisely one bucket; e.g.:

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

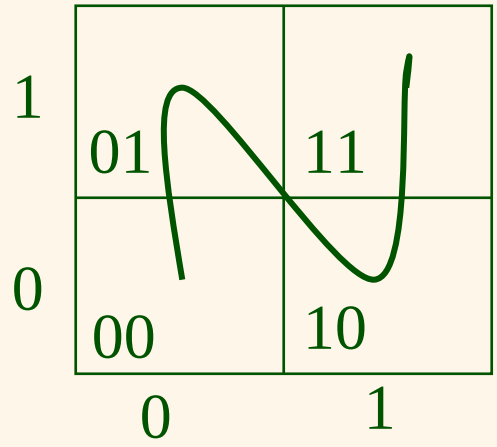


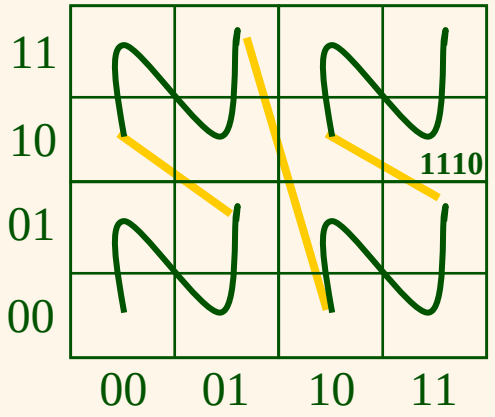
- For rectangle data structures, bucket regions may overlap.

Spatial Indexing by Mapping to 1D Space

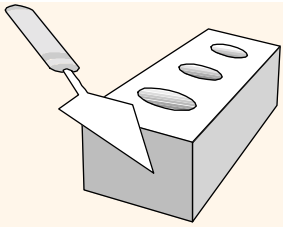


- One dimensional embedding: z-order or bit-interleaving
  - Find a linear order for the cells of the grid while maintaining "locality" (i.e., cells close to each other in space are also close to each other in the linear order)
  - Define this order recursively for a grid that is obtained by hierarchical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

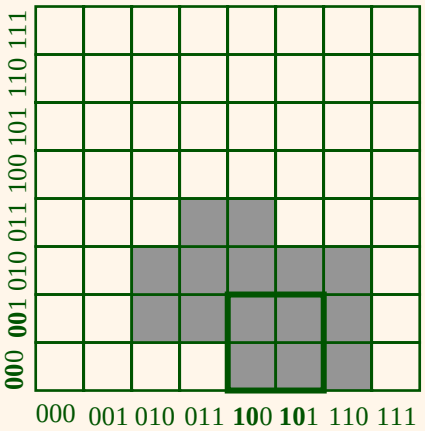


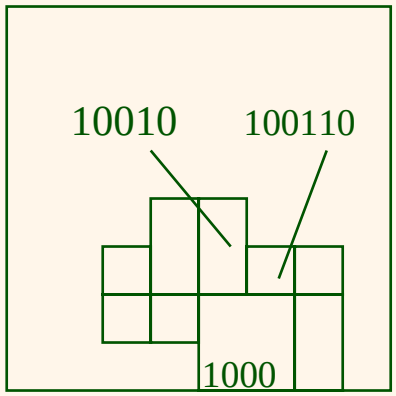


Use Case



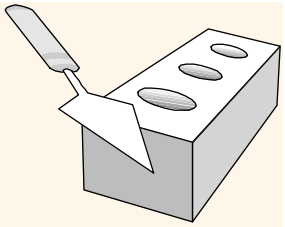
- Any shape (approximated as set of cells) over the grid can now be decomposed into a <u>minimal</u> number of cells at different levels, always using the <u>coarsest</u> possible level



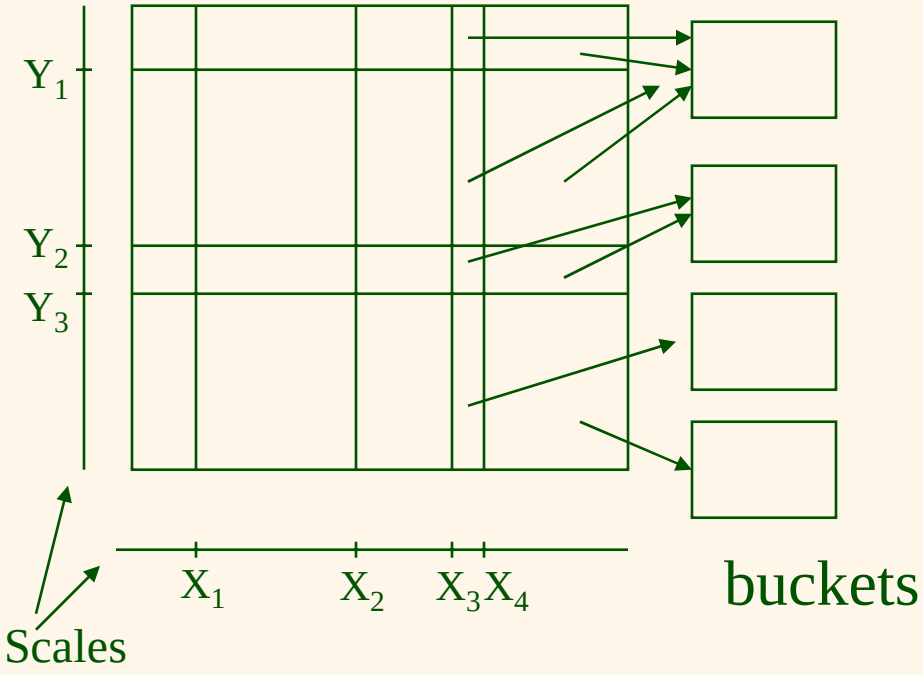


- Hence, for each spatial object, we can obtain a set of "spatial keys"
- Index: can be a B-tree of lexicographically ordered list of the union of these spatial keys

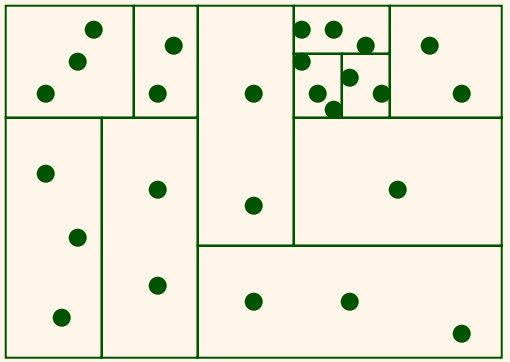
Dedicated Spatial Indexes: Point Indexing



- Spatial index structures for points:

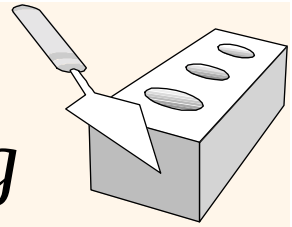


Grid-file



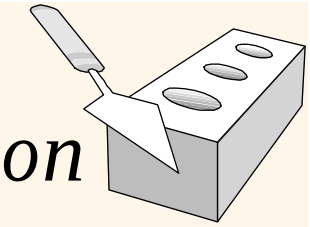
KD-Tree

Dedicated Spatial Indexes: Region Indexing

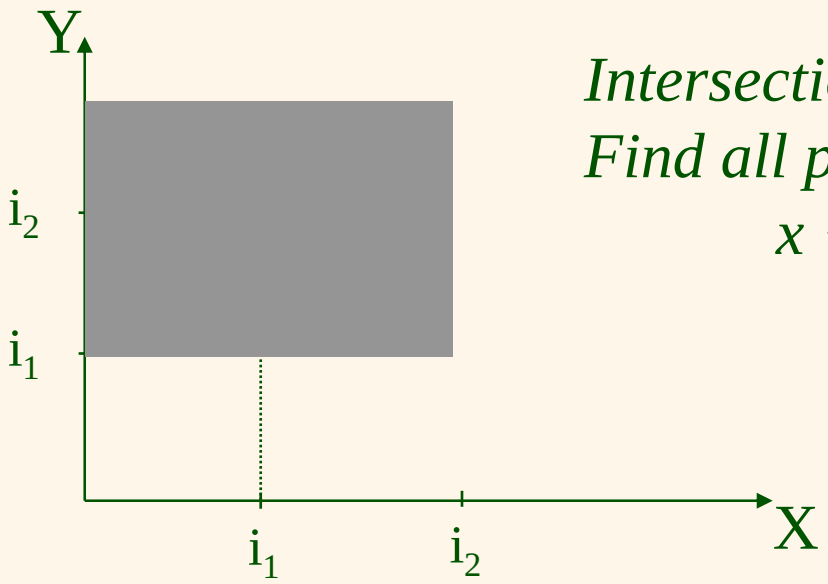


- Spatial index structures for regions:
  - Unlike points, regions / rectangles don't fall into a unique cell of a partition and might intersect partition boundaries
  - Solutions:
    - Transformation: instead of k-dimensional rectangles, 2k-dimensional points are stored using a point data structure
    - 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 of Regions by Transformation

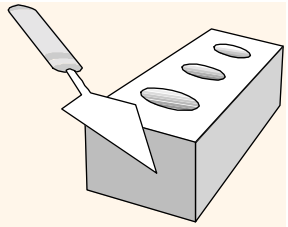


- E.g., a rectangle with 4 coordinates $(X_{left}, X_{right}, Y_{bottom}, Y_{top})$ can be considered as a point in 4D-space
- E.g., consider how an interval $i = (i_1, i_2)$ with 2 coordinates can be mapped to 2D-space (as a point):



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

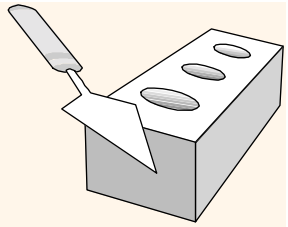
Spatial Join Algorithms



- Filtering the Cartesian product is expensive.
- Depending on the object approximation method used, there are two types of algorithms for spatial join:
  1. Algorithms assuming Grid Approximation
  2. Algorithms assuming Bounding Box Approximation

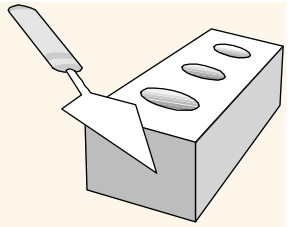
- A join algorithm assuming Grid Approximation with z-order indexing (for the overlap predicate):
  - A parallel scan of two sets of z-elements corresponding to two sets of spatial objects is performed
  - Too fine a grid, too many z-elements per object (inefficient)
  - Too coarse a grid, too many "false hits" in a spatial join

Spatial Join Algorithms (cont'd)



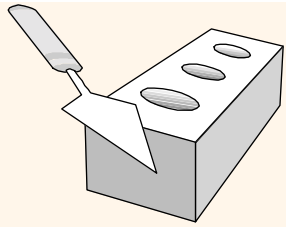
- Join algorithms assuming Bounding Box Approximation:
  - Problem: "For two sets of rectangles R and S, find all pairs *(r,s), r in* R, *s in* S, such that *r* intersects *s.*"
  - Solutions depending on availability of index:
    - **No spatial index on R and S**: *bb_join* which uses a computational geometry algorithm to detect rectangle intersection
    - **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
  - Spatial data types & operations within data definition and query language
  - User interface extensions to handle graphical representation and input of SDT values

System Architecture (cont'd)



- The only clean way to accommodate these extensions is an integrated architecture based on the use of an extensible DBMS; current commercial solutions are OR-DBMSs:
  - NCR Teradata Object Relational (TOR)
  - IBM DB2 (spatial extender)
  - Informix Universal Server (spatial datablade)
  - Oracle 10g (spatial cartridge)
- 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)