

Part5

An Introduction to Spatial Databases

- ▶ Introduction
- ▶ Modeling spatial concepts
- ▶ Organizing the underlying space
- ▶ Spatial data types, algebras, and relationships
- ▶ Integrating geometry into DBMS data model
- ▶ Querying spatial databases
- ▶ Spatial indexes
- ▶ System architecture

Literature and Acknowledgments

- ▶ R.H. Güting: An introduction to spatial database systems. *VLDB Journal* 3:357–399 (1994)
- ▶ These slides are based on slides from
 - ▶ Johann Gamper, Free University of Bozen-Bolzano, Italy

Introduction/1

- ▶ The **goal** of spatial database research is to extend DBMS data models and query languages to be able to manage geometries/geometric objects.
 - ▶ Data structures for geometric shapes
 - ▶ Algorithms for performing geometric computations
 - ▶ Indexing techniques for multi-dimensional space
 - ▶ Extensions of query optimizers
- ▶ The main **motivation** for spatial databases in the past was to support geographic information systems (GIS)
 - ▶ Early GIS made limited use of DBMS technology
 - ▶ Only non-spatial data were stored in a DB; geometries were managed in separate files
 - ▶ Modern GIS systems are built on top of DBMS
 - ▶ All major commercial systems offer spatial extensions (e.g., Oracle, IBM DB2, Informix)

Introduction/2

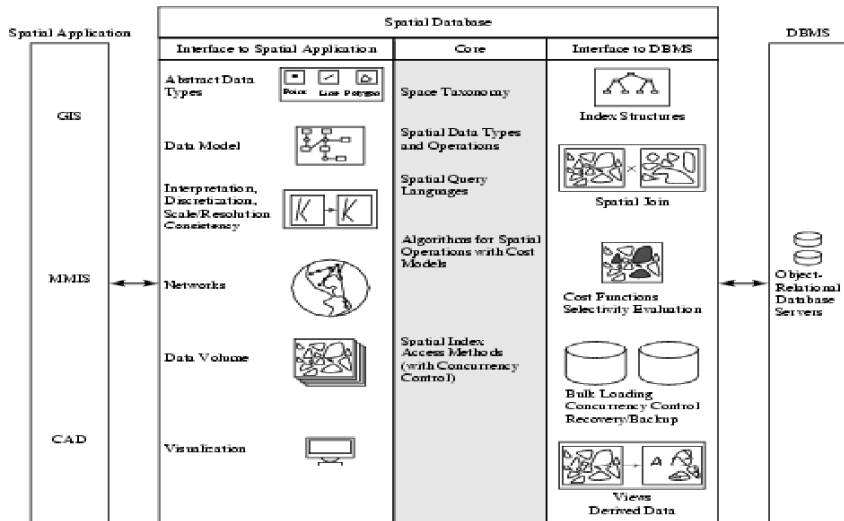
- ▶ Spatial databases have a **wider scope** and are able to represent other spaces (beside geographical spaces in GIS systems)
 - ▶ The layout of a VLSI design
 - ▶ A 3D model of the human body
 - ▶ A protein structure studied in molecular biology
- ▶ An important distinction has to be made between **image databases** and **spatial databases**
 - ▶ Image databases manage raster images of space
 - ▶ Spatial databases manage objects (with clear location and extent) in space
 - ▶ Feature extraction can be used to identify spatial entities within an image that can be stored in a spatial database.

► Definition: A spatial DBMS

- is a DBMS with additional capabilities for handling spatial data
- offers **spatial data types** in its data model and query language
 - Structure in space, e.g., point, line, region
 - Relationships among them, e.g., intersection
- supports spatial data types in its implementation
 - Providing at least **spatial indexing** (retrieving objects in particular area without scanning the whole space)
 - Efficient algorithms for **spatial join** (not simply filtering the Cartesian product)

Introduction/4

► 3-Layer Architecture of a Spatial DBMS

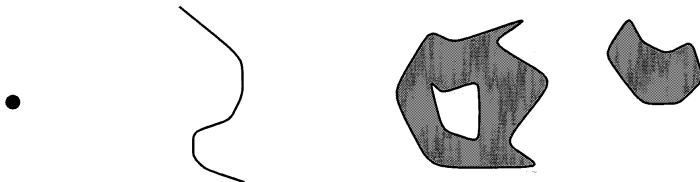


Modeling Spatial Concepts/1

- ▶ The entities to be represented in a spatial database include anything that might appear on a paper map
- ▶ Two alternative views about what needs to be represented
 - ▶ **Objects in space:** Distinct entities arranged in space, each of which has its own geometric description
 - ▶ e.g., cities, rivers, highway networks, forests, etc.
 - ▶ **Space:** Space itself, i.e., to say something about every point in space
 - ▶ e.g., thematic maps, land use, partition of a country into districts
- ▶ To model these diverse entities the following classes of concepts are distinguished
 - ▶ **Single objects**
 - ▶ **Spatially related collections of objects**

Modeling Spatial Concepts/2

- ▶ Three fundamental abstractions of **single objects**
 - ▶ **Point:** An object for which only its location but not its extent is relevant
 - ▶ e.g., cities on a large-scale map, hospitals, subway stations
 - ▶ **Line (curve):** An entity moving through space or a connection in space
 - ▶ e.g., rivers, highways, telephone cables
 - ▶ **Region:** An entity that has an extent in the 2D space
 - ▶ e.g., countries, forests, lakes
 - ▶ May in general have holes and consist of several disjoint pieces



Modeling Spatial Concepts/3

- ▶ The two most important abstractions for **spatially related collections of objects**

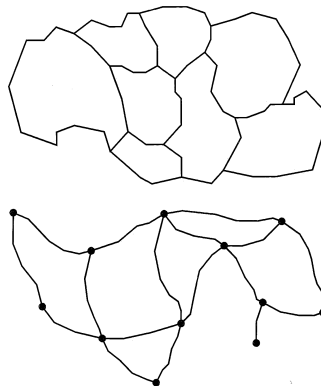
- ▶ **Partition:** A set of region objects that are required to be disjoint

- ▶ The adjacency relationship is of particular interest, i.e., common boundary of region objects
 - ▶ e.g., land use, districts, land ownership

- ▶ **Network:** A graph embedded into the plane, consisting of a set of point objects forming its nodes and a set of line objects describing the geometry of the edges

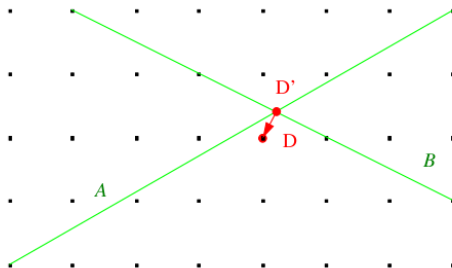
- ▶ e.g., highways, public transport, power supply lines

- ▶ Other interesting abstractions include **nested partition**, e.g., a country partitioned into provinces partitioned into districts, etc.



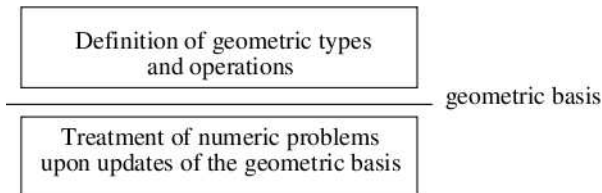
Organizing the Underlying Space/1

- ▶ Euclidean geometry is not suitable as a basis for modeling in spatial databases
 - ▶ Euclidean space is **continuous**, i.e., points are represented by a pair of real numbers $p = (x, y) \in \mathbb{R}^2$
 - ▶ But computer numbers are **discrete**
 - ▶ $\Rightarrow$ space is represented by a discrete raster
- ▶ **Example:** Intersection of two lines
 - ▶ Intersection point is rounded to the nearest grid point
 - ▶ A subsequent test to determine whether the intersection point is on one of the lines yields a false result



Organizing the Underlying Space/2

- ▶ Goal: Avoid computation of new intersection points within geometric operations.
- ▶ **Solution:** Definition of a **discrete geometric basis** for modeling as well as for implementation.



- ▶ Two approaches for a geometric basis:
 - ▶ **Simplicial complexes** (based on combinatorial topology) (Frank & Kuhn 86, Egenhofer, Frank & Jackson 89)
 - ▶ **Realms** (Güting & Schneider 93, Schneider 97)

Organizing the Underlying Space/3

► Basic concepts of **simplicial complexes**

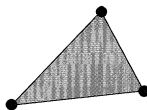
- **d-Simplex:** A minimal object of dimension d
 - Consists of $d + 1$ simplices of dimension $d - 1$



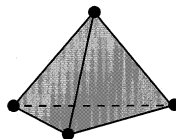
0-simplex



1-simplex



2-simplex

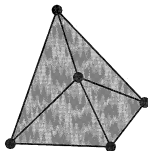


3-simplex

- **Simplicial complex:** A finite set of simplices such that the intersection of any two simplices in the set is a component (face) used in the simplices.
 - c, d, e are not valid complexes



(a)



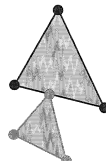
(b)



(c)



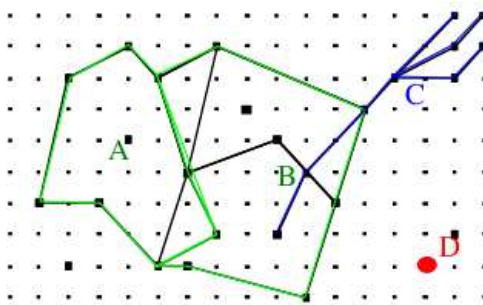
(d)



(e)

Organizing the Underlying Space/4

- ▶ **Realm:** A finite set of points and line segments defined over a grid such that
 - ▶ each point or end point of a segment is a grid point
 - ▶ each end point of a segment is also a point of the realm
 - ▶ no realm point lies within a segment
 - ▶ no two distinct segments intersect except at their end points
- ▶ A realm provides a complete description of the geometry (all points and lines) of an application.



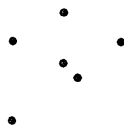
- ▶ Realm elements are used to construct objects (e.g., A, B, C, D)

Spatial Data Types and Algebras

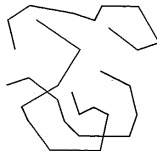
- ▶ The basic spatial abstractions can be embedded into an existing DBMS data model by using abstract data types
- ▶ **Spatial (abstract) data types** encapsulate
 - ▶ the structure of a spatial object, e.g., region
 - ▶ and operations on it, e.g., predicates, functions, construction operators
- ▶ **Spatial algebra:** A collection of spatial data types with related operations
 - ▶ Important properties of an algebra are closure under operations and completeness

ROSE Algebra/1

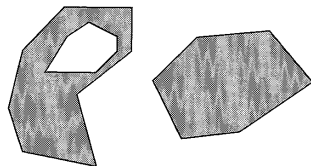
- ▶ **ROSE Algebra** (RObust Spatial Extension, Güting & Schneider 95)
 - ▶ A spatial algebra with realm-based spatial data types (i.e., objects composed from realm elements)
- ▶ **ROSE data types:** points, lines, regions



a points value

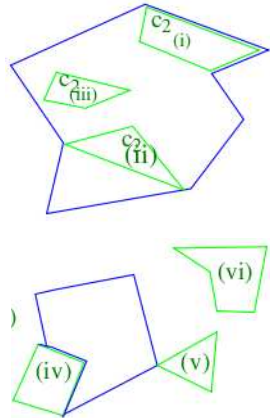


a line value



a region value

- ▶ More complex structure of objects are treated in definitions
 - ▶ c_2 is (area-)inside c_1 (i, ii, iii)
 - ▶ c_2 is edge-inside c_1 (ii, iii)
 - ▶ c_2 is vertex-inside c_1 (iii)
- ▶ c_2 is area-disjoint c_1 (iv, v, vi)
- ▶ c_2 is edge-disjoint c_1 (v, vi)
- ▶ c_2 is vertex-disjoint c_1 (vi)



ROSE Algebra/3

- ▶ **ROSE operations:** ROSE provides precisely defined operations
 - ▶ Let $GEO = \{points, lines, regions\}$, $EXT = \{lines, regions\}$
and $geo \in GEO$, $ext \in EXT$
- ▶ Spatial predicates for topological relationships

inside :	$geo \times regions$	$\rightarrow bool$
intersect, meets :	$ext \times ext$	$\rightarrow bool$
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 :	$regions$	$\rightarrow lines$

ROSE Algebra/4

- ▶ Spatial operators returning numbers

dist : $geo \times geo \rightarrow real$
perimeter, area : $regions \rightarrow real$

- ▶ Spatial operations on set of objects

sum : $set(obj) \times (obj \rightarrow geo) \rightarrow geo$

- ▶ **sum** is a spatial aggregate function: Takes a set of objects together with a spatial attribute of the objects of type *geo* and returns the geometric union of all attribute values
- ▶ e.g., form the union of a set of provinces to determine the area of a country

closest : $set(obj) \times (obj \rightarrow geo) \times geo \rightarrow set(obj)$

- ▶ The **closest** operator determines within a set of objects those whose spatial attribute value has minimal distance from some other geometric object

Spatial Relationships/1

- ▶ **Spatial relationships** (predicates) are the most important operations offered by a spatial DBMS
 - ▶ **Topological relationships:** adjacent, inside, disjoint, etc.
 - ▶ **Direction relationships:** above, below, north_of, etc.
 - ▶ **Metric relationships:** distance < 100
- ▶ Enumeration of all possible topological relationships between two simple regions (no holes, connected)
 - ▶ Based on comparing two objects boundaries (δA) and interiors ($\circ A$)
 - ▶ 8 of the 16 possible combinations are not valid

$\delta A_1 \cap \delta A_2$	$\delta A_1 \cap \circ A_2$	$\circ A_1 \cap \delta A_2$	$\circ A_1 \cap \circ A_2$	Relationship name
$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	A_1 disjoint A_2
$\emptyset$	$\emptyset$	$\neq \emptyset$	$\neq \emptyset$	A_2 in A_1
$\emptyset$	$\neq \emptyset$	$\emptyset$	$\neq \emptyset$	A_1 in A_2
$\neq \emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	A_1 touch A_2
$\neq \emptyset$	$\emptyset$	$\emptyset$	$\neq \emptyset$	A_1 equal A_2
$\neq \emptyset$	$\emptyset$	$\neq \emptyset$	$\neq \emptyset$	A_1 cover A_2
$\neq \emptyset$	$\neq \emptyset$	$\emptyset$	$\neq \emptyset$	A_2 cover A_1
$\neq \emptyset$	$\neq \emptyset$	$\neq \emptyset$	$\neq \emptyset$	A_1 overlap A_2

Spatial Relationships/2

- ▶ Extensions have been proposed
 - ▶ Consider dimension of the intersection → 256 combinations out of which 52 are valid
- ▶ **Minimal solution:**
 - ▶ 5 basic relationships: *touch*, *in*, *cross*, *overlap*, *disjoint*
 - ▶ 3 operators *b*, *f*, and *t* to get the boundaries
 - ▶ *b* applied to an area returns the boundary line
 - ▶ *f* and *t* return the end points of a line
- ▶ Properties of the minimal solution:
 - ▶ 5 relationships are mutually exclusive
 - ▶ 5 relationships + 3 boundary operators can distinguish all 52 valid configurations above

Integrating Geometry into the DBMS Data Model

- ▶ Spatial datatypes can be embedded in any data model, e.g., the relational data model
- ▶ DBMS data model must be extended with SDTs at the level of atomic data types (integer, string)
- ▶ **Basic idea**
 - ▶ Represent **spatial objects** by objects (of the DBMS data model) with **at least one SDT attribute**.
 - ▶ Relational data model: spatial objects are tuples with at least one SDT attribute
 - ▶ **Example:** Relational tables to store cities, rivers, and countries

```
cities( name:string, population:int, location:points )
rivers( name:string, route:line )
highways( name:string, route:line )
states( name:string, area:region )
```
 - ▶ Representation of **spatially related collections of objects** (e.g., partitions) as a set of objects with a region attribute
 - ▶ Loses some information, e.g., regions are disjoint

Querying Spatial Databases/1

- ▶ Connect operations of a spatial algebra to the facilities of a DBMS query language
- ▶ Two main issues must be considered
 - ▶ Fundamental operations for manipulating sets of database objects
 - ▶ spatial selection, spatial join, etc.
 - ▶ Graphical input and output

Querying Spatial Databases/2

- ▶ **Spatial selection:** Select those objects that satisfy 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 km from Hagen?*

```
select cname
from cities c
where dist(c.center,Hagen.center) < 100 and c.pop > 500k
```

Querying Spatial Databases/3

- ▶ **Spatial join:** 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 km?*

```
select r.rname, c.cname  
from rivers as r, cities as c  
where r.route intersects Bavaria.area and  
dist(r.route,c.area) < 50 km
```
- ▶ *What is the total population of cities in France?*

```
select sum(c.pop)  
from cities as c, states as s  
where c.location inside s.area and s.name = 'France'
```
- ▶ *Make a list, showing for each country the number of its neighbor countries?*

```
select s.name, count(*)  
from states as s, states as t  
where s.area adjacent t.area  
group by s.name
```

Querying Spatial Databases/4

- ▶ **Spatial function application:** Apply (spatial) functions to each member of a set.
- ▶ *Return the part of river Rhine that is within Germany?*

```
select intersection(r.route, s.area)
from rivers as r, states as s
where r.name = 'Rhine' and s.name = 'Germany'
```

Querying Spatial Databases/5

► Graphical input and output

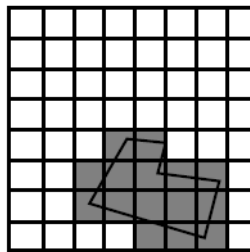
- While traditional DBMS deal with alphanumeric data (types), some data (types) in spatial DBMS require a graphical representation
 - Input: How to determine “window” or “Bavaria” in the previous examples?
 - Output: How to show “intersection(route,Bavaria.area)” or “r.route”?
- The final information to be retrieved is often the result of several queries, the result of which should be graphically overlayed on top of a map
 - Graphical combination of several query results (e.g., add/remove layers, change order of layers)
 - Display of context (e.g., show background such as a raster image or boundary of states)

Spatial Indexing/1

- ▶ Mainly used to support spatial selection
 - ▶ but supports also other operations, e.g., spatial join or finding the closest object
- ▶ A spatial index 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:
 - ▶ Map spatial objects to a 1-D space and utilize standard indexing techniques, e.g., B-tree
 - ▶ Dedicated spatial index data structures, e.g., R-tree

Spatial Indexing/2

- ▶ A fundamental idea of spatial indexing is the use of **approximations**
 - ▶ Continuous approximation, e.g., a bounding box
 - ▶ Grid approximation



- ▶ Leads to a **filter and refine** strategy for query processing
 1. Filter: Returns a set of candidate objects which is a superset of the objects fulfilling a predicate
 2. Refine: For each candidate, the exact geometry is checked

Spatial Indexing/3

- ▶ Most spatial data structures are designed to either store **points** (for point values) or **rectangles** (for line and region values)
- ▶ Operations on those structures: insert, delete, check membership
- ▶ Typical 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
 - ▶ for rectangles:
 - ▶ *Intersection query*
 - ▶ *Containment query*

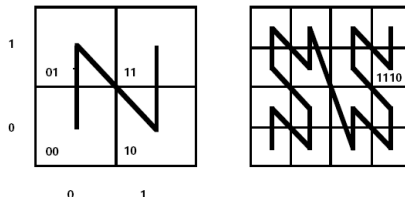
1-D Embedding of Grid Approximation/1

► 1-D embedding of grid approximations: basic idea

1. Find a linear order for the cells of the grid such that cells close together in space are also close to each other in the linear order
 - Maintains locality
2. To define this order recursively for a grid that is obtained by a hierarchical subdivision of space

1-D Embedding of Grid Approximation/2

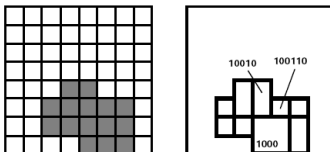
- ▶ **Bit-interleaving** is the most popular such order (Morton 1966)
 - ▶ Later also called **z-order** (Orenstein and Manola, 1988)



- ▶ Each cell at each level of the hierarchy has an associated bit string whose length corresponds to the level to which the cell belongs.
 - ▶ e.g., the top-right cell in the left diagram has bit string 11, on the right-side cell 1110 is shown.
 - ▶ The bit-string 1110 is obtained by choosing 11 at the top level, and then 10 within the top level quadrant.
- ▶ The order which is imposed on all cells of a hierarchical subdivision is given by the *lexicographical order of the bit strings*.

1-D Embedding of Grid Approximation/3

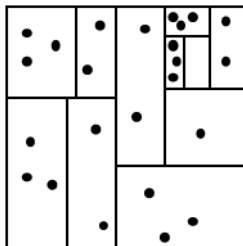
- ▶ Any shape (approximated as a set of cells) over the grid can now be decomposed into a **minimal number** of cells at different levels (always using the highest possible level).
 - ▶ It can therefore be represented by a set of bit strings, called **z-elements**



- ▶ For a spatial object, the corresponding set of z-elements builds a set of spatial keys
- ▶ **Spatial index:** Put z-elements as spatial keys in lexicographical order into a B-tree (cf., UB tree).
- ▶ Due to the *proximity-preserving property* various types of queries can be answered relatively efficiently, e.g., containment or range query with rectangle r
 - ▶ determine z-elements of r
 - ▶ for each z-element z scan a part of the leaf sequence of the B-tree having z as prefix.
 - ▶ Check these candidates for actual containment.

Spatial Index Structures

- ▶ A (dedicated) spatial index structure organizes objects 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
 - ▶ the space is partitioned and each point belongs to precisely one bucket
 - ▶ e.g., a kd-tree partitioning of 2d-space where each bucket can hold up to 3 points



- ▶ For rectangle data structures the bucket regions may overlap

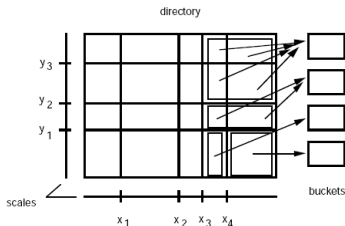
Spatial Index Structures for Points/1

► Spatial index structures for points

- Data structures of representing points in k dimensions (multi-attribute) have a long tradition, e.g., a tuple $t = (x_1, \dots, x_k)$
- Can be used to store geometrical points

► **GRID index:** Spatial index structure for points (Nievergelt, Hinterberger, and Sevcik 84)

- The following example partitions the data space into *cells* by an irregular grid

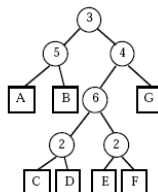
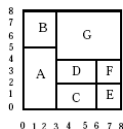


- The directory is a k -dimensional array whose entries are logical pointers to buckets.
- All points in a cell are stored in the bucket pointed to by the corresponding directory entry.
- The scales are small and are kept in main memory; the directory is on the disk.

Spatial Index Structures for Points/2

► **kd-Tree** (Bentley 75)

- Binary tree where each internal node contains a key drawn from one of the k dimensions
- The key in the root node (level 0) divides the data space with respect to dimension 0, the keys in its sons (level 1) divide the two subspaces with respect to dimension 1, and so forth, up to dimension $k - 1$, after which cycling through the dimensions restarts.
- Leaves contain the points to be stored



x
y
x
y

- **KDB-tree** (Robinson 81): introduce buckets, paginate the binary tree, all leaves at the same level (like B-tree)
- **LSD-tree** (Henrich et al. 89): abandon strict cycling through dimensions; clever paging algorithm keeps external path length balanced even for very unbalanced binary trees.

Spatial Index Structures for Rectangles/1

- ▶ **Spatial index structures for rectangles**

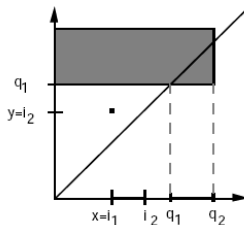
- ▶ Unlike points, rectangles do not fall into a unique cell of a partition and might intersect partition boundaries
- ▶ Three main approaches:
 - ▶ Transformation approach
 - ▶ Overlapping bucket regions
 - ▶ Clipping

Spatial Index Structures for Rectangles/2

► Transformation approach

- Idea: Consider simple geometric shapes as points in higher dimensional space (the parameter space)
- k -dimensional rectangles are transformed into $2k$ -dimensional points, and a point data structure is used.
- Rectangle (x_l, x_r, y_b, y_t) can be viewed as a point in 4-D space

► **Example:** Interval $i = [i_1, i_2]$ is mapped into a point (x, y) in 2-D space

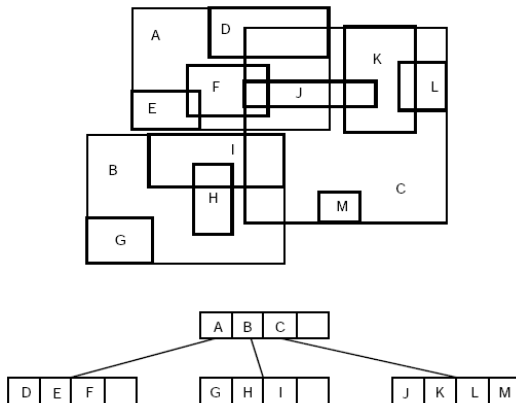


- An intersection query with an interval $q = [q_1, q_2]$ translates to a condition: Find all points (x', y') s.t. $x' < q_2$ and $y' > q_1$.
- All intervals intersecting q are in the shaded area

Spatial Index Structures for Rectangles/3

► Overlapping bucket regions

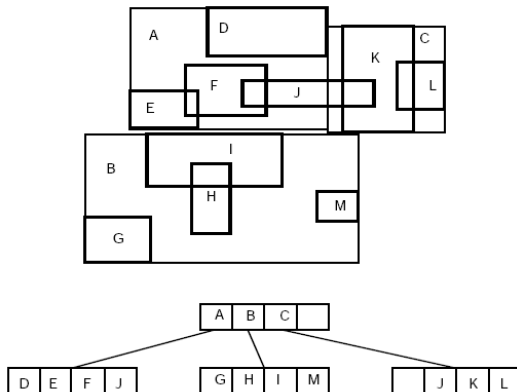
- Partitioning space is abandoned and bucket regions may overlap, e.g., **R-tree** (Guttman 84)
- Advantage: Spatial object (or key) is in a single bucket
- Disadvantage: Multiple search paths due to overlapping bucket regions



Spatial Index Structures for Rectangles/4

► Clipping

- Bucket regions are disjoint, but data rectangles are cut into several pieces (if necessary), e.g., **R⁺-tree** (Sellis, Rossopoulos and Faloutsos 87)
- Advantage: Less branching in search
- Disadvantage: Multiple entries for a single spatial object



Spatial Join/1

- ▶ Very active research area in the last few years
- ▶ Traditional join methods such as hash join or sort/merge join are not applicable
- ▶ Filtering Cartesian product is expensive
- ▶ Proposed methods can be classified along the following criteria:
 - ▶ Grid approximation/bounding box
 - ▶ None/one/both operands are represented in a spatial index structure
- ▶ **Grid approximations** with an overlap predicate
 - ▶ A parallel scan of two sets of z-elements corresponding to two sets of spatial objects is performed
 - ▶ Similar to a merge join

Spatial Join/2

- ▶ **Bounding boxes:** For two sets of rectangles R and S all pairs (r, s) , $r \in R$ and $s \in S$ such that r intersects s :
 - ▶ *No spatial index on R and S :* rectangle intersection algorithm uses a computational geometry algorithm to detect rectangle intersection; external divide and conquer algorithm that is similar to external merge sorting
 - ▶ *Spatial index on either R or S :* index join scans 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 (efficient if non-indexed operand is not too big)
 - ▶ *Both R and S are indexed:* synchronized traversal of both structures so that pairs of cells of their repesctive partitions covering the same part of space are encountered together.

System Architecture/1

► Requirements

- Representations for the data types of a spatial algebra
- Procedures for atomic operations (e.g., overlap)
- Spatial index structures
- Access operations for spatial indices
- 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 and operations within data definition and query language
- User interface extensions to handle graphical representation and input of SDT values

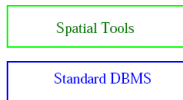
System Architecture/2

▶ **GIS architectures using a closed DBMS**

- ▶ First generation systems: built on top of file system
 - ▶ no high level data definition
 - ▶ no flexible querying
 - ▶ no transaction management
 - ▶ ...
- ▶ Using a standard (mostly relational) DBMS
 - ▶ Layered architecture
 - ▶ Dual architecture

System Architecture/3

► Layered architecture



► Representation of SDT values:

- Decompose SDT value into a set of tuples, one tuple per point or line segment
- Obviously a terrible solution

states	sid	sname	pop
	s01	"Bavaria"	7 000 000

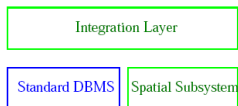
edges	sid	x1	y1	x2	y2	edge
	s01	134.78	92.514	137.13	93.84	1
	s01	137.13	93.84	139.11	96.37	2
	s02					
	...					

► Represent SDT values in “long fields” of DBMS

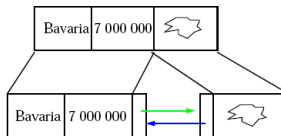
- DBMS handles geometries only as uninterpreted byte strings

System Architecture/4

► Dual architecture



- SDT representation broken into two pieces:



- Advantage: Freedom to use efficient data structures and algorithms in the spatial subsystem
- Problems: Query must also be decomposed into two parts
 - Complex query processing
 - No global query optimization possible

► **Integrated spatial DBMS architecture**

- Using an Extensible DBMS
- The only clean way to accommodate the requirements for spatial databases
- Leads to an integrated architecture such that
 - No difference in principle between a standard data type (STRING) and a spatial data type (REGION); same for operations
 - No difference in principle between index for standard attribute (e.g., B-tree) and for spatial attribute (R-tree)
 - Sort/merge join and bounding-box join are basically the same
 - Mechanisms for query optimization do not distinguish spatial or other operations

System Architecture/6

- ▶ Spatial DBMS prototypes built on extensible systems
 - ▶ SECONDO (Güting 04)
 - ▶ PostgreSQL with PostGIS
 - ▶ MONET (Boncz et al. 96)
 - ▶ DASDBS (Schek et al. 90)
 - ▶ GRAL (Güting 89)
 - ▶ PROBE (Orenstein 86)
- ▶ Commercial extensible DBMS with spatial extensions
 - ▶ Informix Universal Server (with Geodetic DataBlade)
 - ▶ IBM DB2 (with Spatial Extender)
 - ▶ Oracle Spatial

Summary

- ▶ A spatial DBMS offers capabilities for dealing with spatial data, it offers spatial data types, and it supports spatial data types in its implementation (spatial indexing and spatial join)
- ▶ The most important spatial data types are *points*, *lines* and *regions*.
- ▶ The issues that arise from the precision of geometric operations must be handled in a principled way.
- ▶ Along with spatial data types comes a spatial algebra (i.e., operations on spatial data types) and spatial relationships.
- ▶ Spatial data types must be integrated into the DBMS at the level of standard data types. This is best achieved with an extensible DBMS.
- ▶ In order to query spatial databases we need support for graphical input and output. Spatial indexing techniques are necessary to efficiently answer queries.