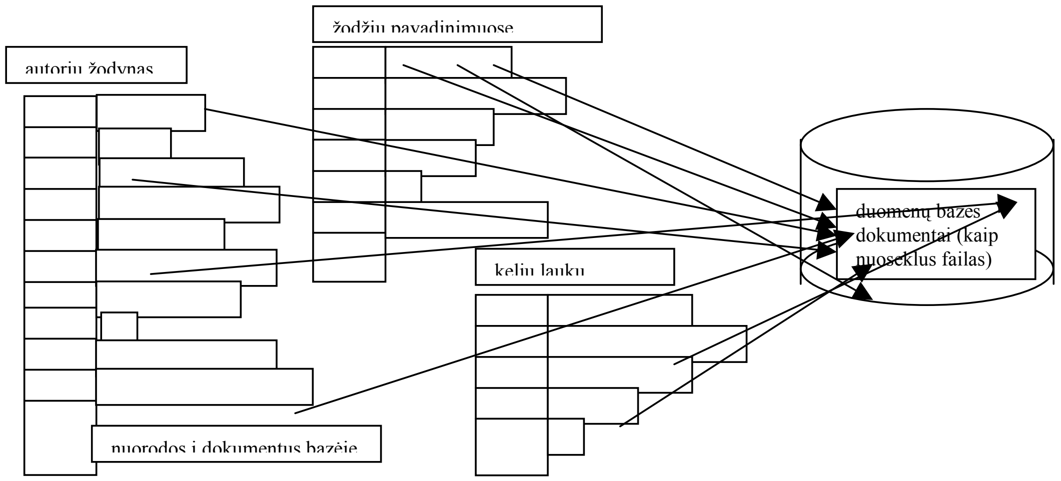
Indices in Information Systems

Algimantas Juozapavičius

algimantas.juozapavicius@maf.vu.lt

Data structures and indices are well-known in traditional (textual or numerical) database systems, as well as in information systems.


žodžiu pavadinimuose
autoriu žodynas
duomenų bazes dokumentai (kaip nuoseklus failas)
keliu lauku
nuorodos į dokumentus bazėje

The logical scheme of indices in a database

Data structures (or abstract data types) are used to design algorithms for data manipulation like sorting, searching, editing of structured data etc. Data structure is a set of data together with a set of well-defined operations and constrains.

The ***indices*** correspond to ***dictionaries and catalogs*** in traditional databases. The dictionaries are used to:

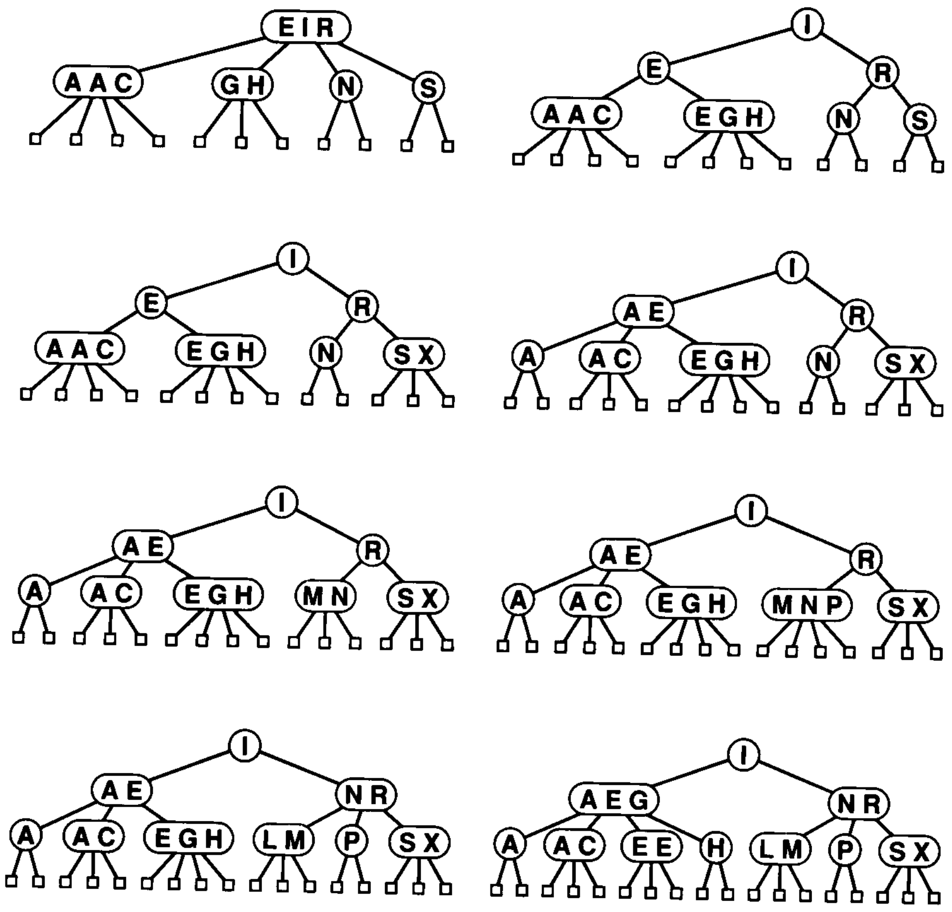
- provide a quick and easy access to data;
- save time and operations while editing, searching, inserting, deleting of data;
- provide additional services while designing queries, analyzing the content of data, etc.

These properties play different roles in databases either in information systems. For instance, the last property is essential in spatial or multimedia IS.

The indices in recent information systems are used to reflect two major aspects of information:

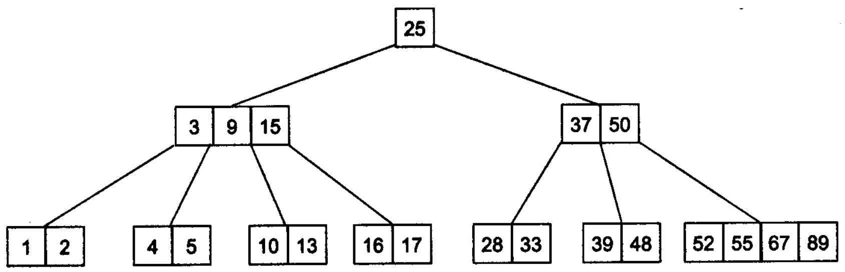
- to organize data in computer memory (especially in secondary and ternary memory) in a efficient way;
- to reflect the most important concepts of information (so-called content-based indices).

There are many data structures serving dictionaries or catalogs, as well as operations on them (the ***data structure*** of dynamic set that supports ***search, insert,*** and ***delete*** operations is called the ***DICTIONARY*** ADT). For instance, the ***2-3-4 trees*** are often used (they are not suitable for operations with data in secondary memory):



2-3-4 tree for “A S E A R C H I N G E X A M P L E”

Another data structure often implemented in many DBMS and well-suited for the manipulations with data in secondary (hard-disk) memory is represented by so-called ***B-trees***. A B-tree of order ***M* = *5*** is shown below:



The B-trees are very efficient:

- for search operations (to index a list of 5 millions of items (words) with a B-tree of order M = 100, the height of the tree would be equal just to 5);
- adjustable to the size of page of hard-disk memory.

Unfortunately, all such data structures have an essential ***drawback***: they work quite well only with ***linearly ordered data***.

The situation has been changed very much with the introduction or invention of hypermedia and multimedia data, as well as with spatial and temporal database systems.

The ***hypermedia*** (nonlinear text), ***multimedia*** (integration of text, images, graphics, sound and video recordings), spatial and temporal data have been changing the notions of data structures and indices drastically.

There were many new data structures suggested, especially for multidimensional, spatial and temporal data – data that can't be considered as a totally sorted set.

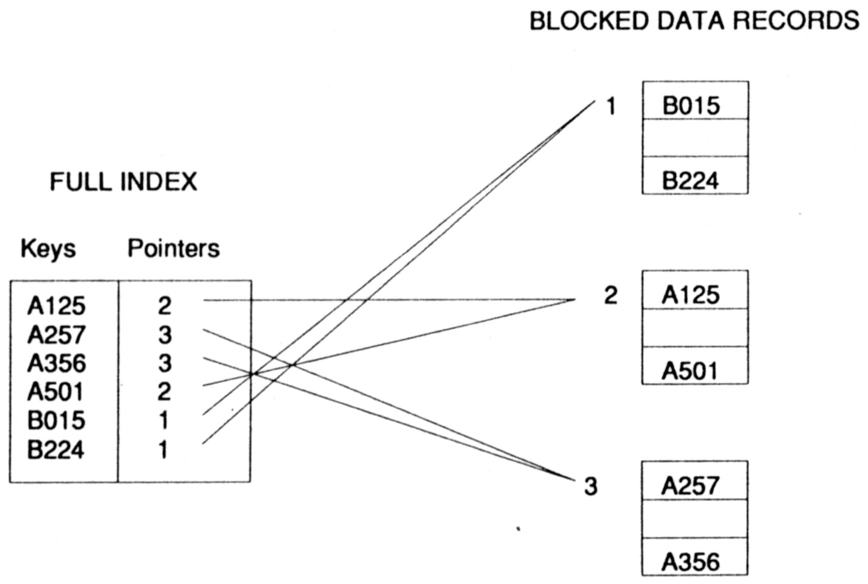
Indices influenced and forced major changes also:

- many nonlinear search strategies appeared;
- indexing techniques have to be applied to graphical, audio, visual data.

Indexing of Spatial Information

The subject of ***spatial indexing*** in a ***database context*** is perhaps the most difficult problem in spatial information systems. The majority of books and commercial products on spatial information systems say little about the topic of indexing, even though data retrieval performances are directly linked to spatial indices.

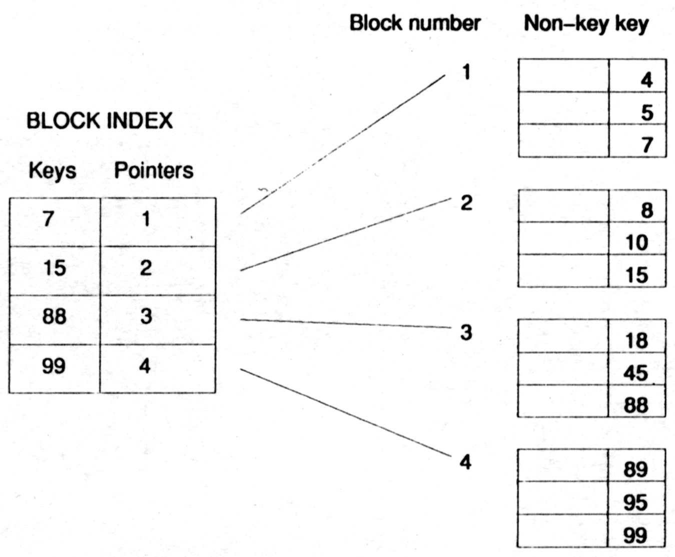
An indexed file provides a way to organize data as opposed to a "simple" list, that is an unordered (except for chronology of creation) arrangement of records. Ordering using alphabetical sequence or numerical identifier sequence, is an improvement over an unordered list for searching quickly using hierarchical binary search techniques:



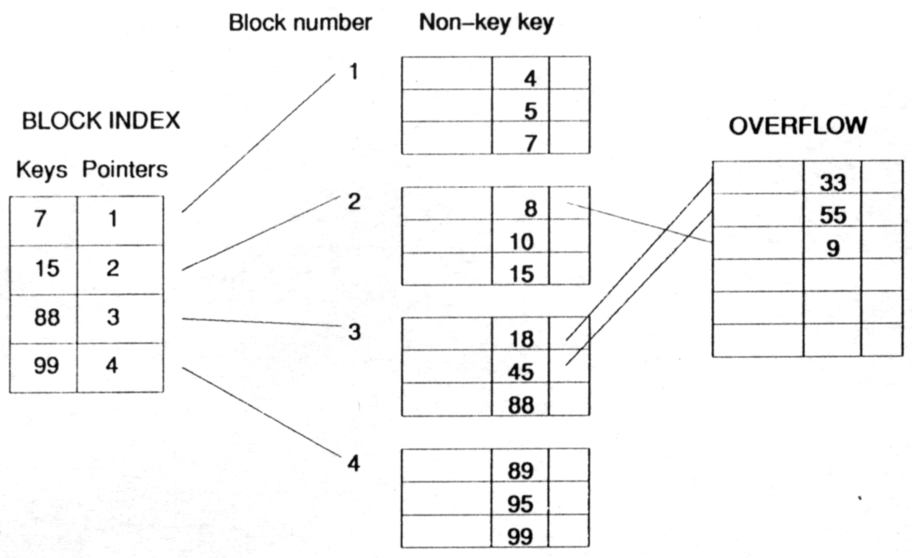
Full index access method

Even so the indexed files may provide more effective access to data because information other than the key identifier is used. However, indexed files are generally more awkward to deal with if a database is changed ***frequently***, for the index itself as well as the data file must be modified, and this operation can be very costly.

At the beginning of computer era there were many attemps to improve or modify file indexing methods. These methods illustrate issues involved in accessing particular pieces of information in a computerized information system, enabling us to understand them.



Accessing by block index

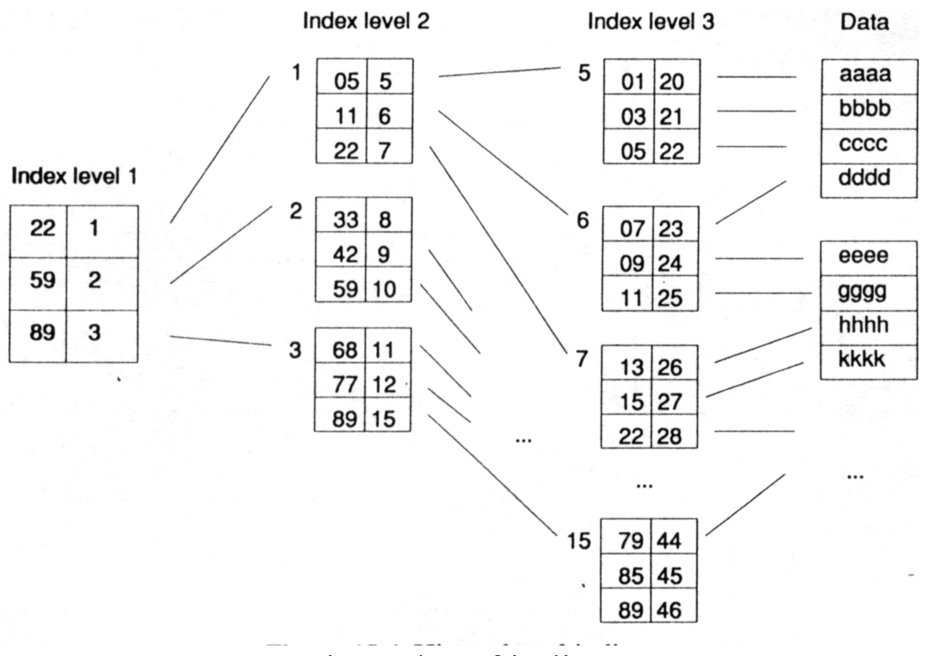


Accessing by block index with overflow zone

The issues essential for accessing particular items of information are:

- the way in which the computer physical encoding is undertaken;
- the design of data structures to facilitate index building;
- the provision of tools by the software systems used for storing and managing data;
- the attributes that are used for building an index.

Attributing disk addresses to keys can be performed according to several possibilities, like sequentially or indexed sequentially, so the former case necessitating the creation of a primary index (the master index) pointing towards several secondary indices. For huge indices, more than two hierarchical levels usually are necessary:



Hierarchy of indices

There may be often ***random access*** imployed, which is a direct access via a special key-to-address transformation, called a ***hashing function.***

Indexing in relational databases

In relational database commercial software products, the user has two **options** for indexing:

1. to do nothing, in which case a product-specific tuple-to-address procedure is provided in order to place the tuples arriving at the physical storage;
2. to create special indices based not only on identifiers (keys) but also on some carefully chosen attributes.

In the SQL type databases, the ***index*** creation using attributes is realized by a **CREATE INDEX** statement (to create an index for Country-name for the relation of POP(Country_name, Population, Capital_city):

CREATE INDEX COUNTRY-INDEX **ON** POP (Country-name)

The combined index such as:

CREATE INDEX DOUBLE-INDEX **ON** POP (Population, Capital-city **DESC)**

can be created. Notice that in this statement a single index with concatenated keys was created. Whenever to have two different indices, two statements should be used:

CREATE INDEX FIRST-INDEX **ON** POP (Population)
CREATE INDEX SECOND-INDEX **ON** POP (Capital_city **DESC)**

This process provides means to accelerate retrieval although prior knowledge is needed in choosing the best attributes:

- there is no reason for building an index using Capital_city if we never need the information for capital city;
- the process will proceed faster if the attribute that discriminates sufficiently well to pick out the smallest number of cases is used as the first index level.

Data retrieval performance depends not only on the technical aspects of the indexing process, but upon the choice of the attribute(s) for making the index. How well this choice is made clearly depends on knowledge of the phenomena.

Moreover, the attributes chosen for building indices may need to vary depending on purpose, and will almost certainly vary with differing views of a database contents by different users. In this regard it is an important technique to separate the index from the items being indexed.

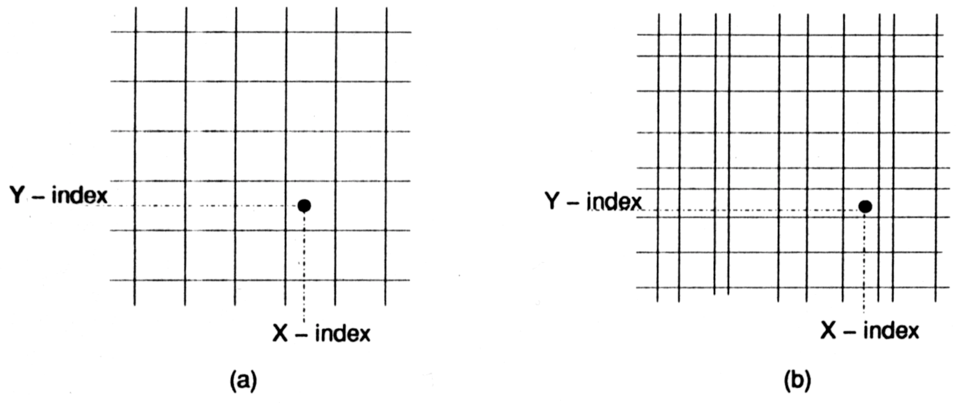
So, there must be a sensitivity to both selection of data items to be used for building an index and to the physical organization of them.

Spatial Indexing

The role of spatial indexing is to accelerate the retrieval of information based on location, especially for large databases. An access mechanism can be as basic as a ***geographic name*** or as involved as a concatenated

numerical code made up of several parts. However, a spatial index should provide an access path to a location or a block of earth space, not necessarily directly to a particular object.

A first possibility is to superimpose a grid, having squares or rectangles and to use the coordinates for an object as a spatial access for that object (in ORACLE: fixed-size tile and variable-size tile) as shown in figure:



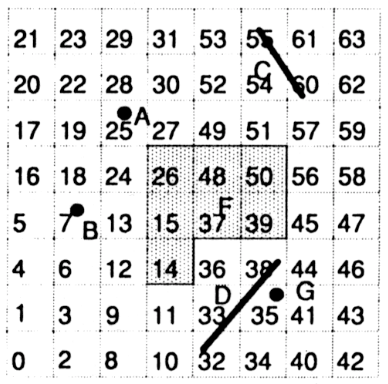
Double indexation: (a) fixed-grid indexing, (b) variable-grid indexing

But the necessity for creating two distinct indices, for the ***x*** and ***y*** orientations, is a substantial disadvantage. That is to say, there is a need to search into those two indices and then combine the results.

Similarly, at three dimensions, there are disadvantages in working with the three necessary indices.

The locators like ***(x, y)*** *or* ***(x, y, z)*** coordinates, expressed with two or three dimensions, cannot be treated in the classical way directly. Indeed, even if the problem of ***multidimensionality*** can be solved by composite indices, there still remains the most important problem to deal with an infinite number of keys.

The next example is to consider the territory encompassing three point objects A, B and G, two line objects C and D, and one area F. It's possible to imagine a spatial index built with Peano keys with the runlength encoding scheme in the object index, that is giving the beginning and ending Peano keys as shown in figure:



SPATIAL INDEX

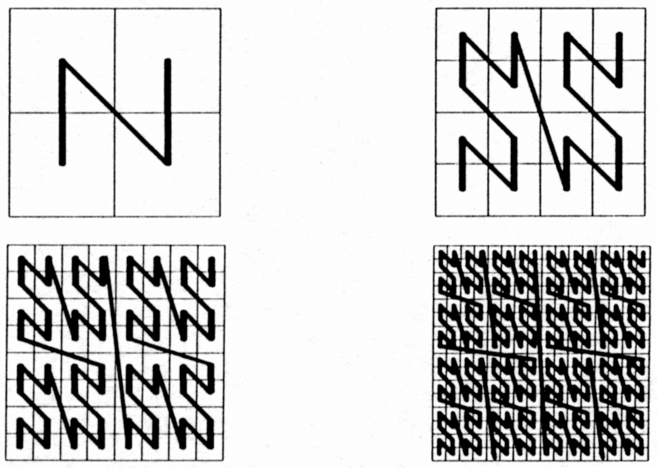
Peano key	Objects
7	B
14	F
15	F
25	A
26	F
32	D
33	D
35	D,G
37	F
38	D
39	F
48	F
50	F
54	C
55	C
60	C

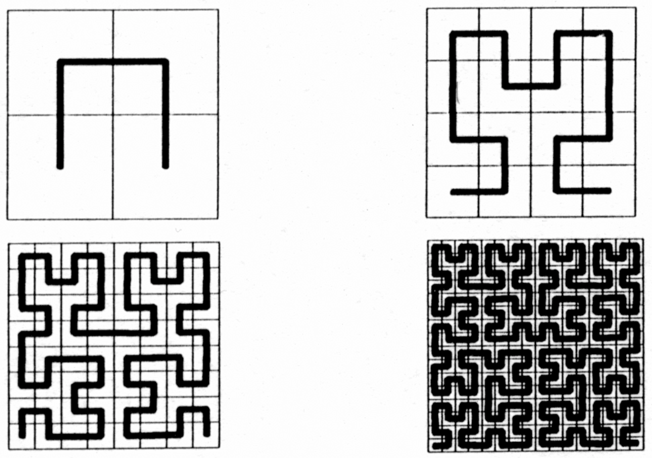
OBJECT INDEX

Objects	Peano keys
A	25–25
B	7–7
C	54–55
C	60–60
D	32–33
D	35–35
D	38–38
F	14–15
F	26–26
F	37–37
F	39–39
F	48–48
F	50–50
G	35–35

(a) (b) (c)

Spatial indexing: (a) map, (b) spatial index, (c) object index


(a) Initial steps of the Peano N curve


(b) Initial steps of the Hilbert curve

Several ways are possible for producing the desirable spatial indices:

1. to consider points as fractal and to order them by a space-filling curve, determining a specific level of resolution with fractal geometry.
2. to construct extents, like minimum bounding rectangles, circles and so on, and classify them into a hierarchy via a valid splitting rule, selecting privileged points in the Euclidean spirit.
3. to use a quadtree mixing Euclidean and fractal geometry.

Indexing by space-filling curves

Space-filling curves, such as the Peano (N order) or Hilbert (Π order), can order all points within the cover by means of the one-dimensional keys, but may not be the preference for all situations.

Moreover, different curves have different properties of ordering, stability or computational simplicity. Selection can thus be made on the basis of several factors:

- A first criterion for evaluating the different types of curve is the ability to provide a spatial reference for every entity. There is no problem here for both curves, although for the Hilbert order, we need to know *a priori* the cover in order to prevent instability.

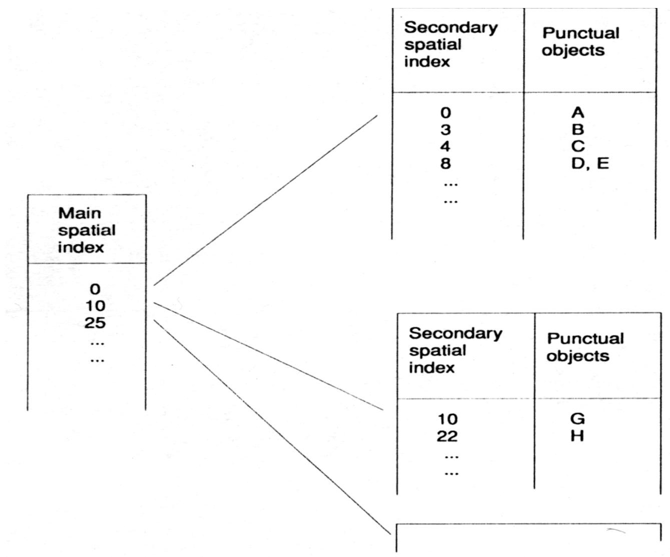
- A second criterion is the facility for passing from one point to its neighbours. Since two neighbouring points in the Hilbert curves are adjacent in the space, this implies that the ***n*** order is a good candidate; this aspect is not always guaranteed in the N order.

- A third criterion is the rapidity of computing keys from coordinates and vice versa. Due to the bit-interleaving procedure, the N order is the quicker ordering out of the Peano and Hilbert curves, and, it is much easier to create keys for the Peano curve then for the Hilbert.

- Another criterion is the utility of spatial indexing in conjunction with quadtrees in order to get hierarchical spatial indexing. In addition, a spatial index must be able to organize punctual, areal and possibly volumic objects.

For points, Peano keys are fine and they can easily be extended to areas in connection with quadtrees. However, for long lines or curve portions, this space-filling curve kind of spatial indexing is not sufficient.

As an illustration of the general process, there can be a first and second level indices for point objects using Peano keys:



Spatial indexing with Peano keys

The index is a hierarchical directory allowing more efficient retrieval than a sequential index, especially when a large number of objects are dealt with. The keys are shown here in decimal form, in practice they are, of course, binary digits.

A spatial index for areal objects can be similarly constructed, although in this case, it is necessary to mention the low and high values for the range of space covered because areal objects are likely to be located in several portions of the curves. So the secondary index shows the low and high value Peano keys.

However, the Peano ordering is not necessarily the most efficient. Recent comparative studies of orderings based on space-filling curves has been undertaken by Abel and Mark (1990), and Faloutsos and Roseman (1989), including a comparison of Peano keys and Hilbert keys. The numerical results of this latter study show that Hilbert indexing is the best for the rapidity of spatial retrieval.

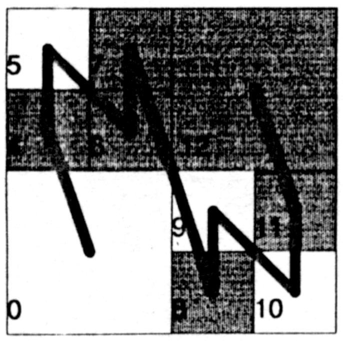
Recall, though, that the creation of the Hilbert keys is difficult, and that they are not stable when the space has to be extended.

The main ***drawbacks*** of this space-filling curve approach is that the keys are sensitive to orientation and to the position of the Cartesian space origin.

Indexing by quadtrees

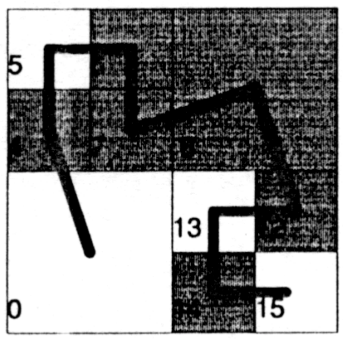
The use of quadtrees is an interesting possibility for spatially indexing objects:

(a) Peano keys



Peano key	Side Length	Colour
0	2	White
4	1	Black
5	1	White
6	1	Black
7	1	Black
8	1	Black
9	1	White
10	1	White
11	1	Black
12	2	Black

(b) Hilbert keys

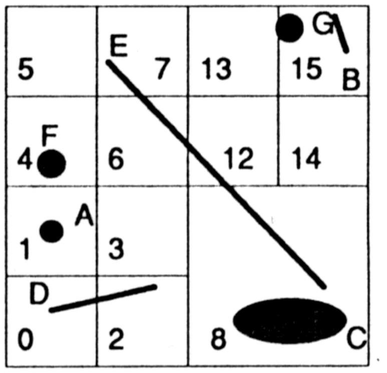


Hilbert key	Side Length	Colour
0	2	White
4	1	Black
5	1	White
6	1	Black
7	1	Black
8	2	Black
12	1	Black
13	1	White
14	1	Black
15	1	White

Spatial index encoding with Peano and Hilbert keys

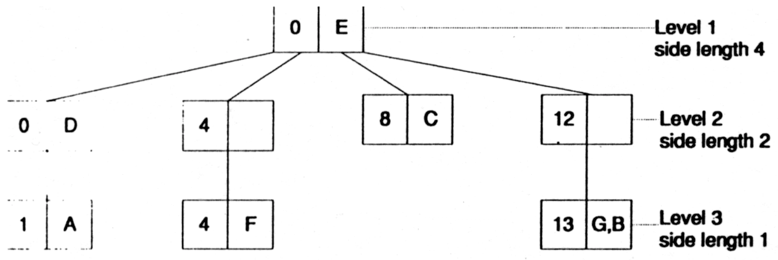
In order to organize several objects in a quadtree, it is taken for each of them its minimum quadrant, that is the smallest entire square bounding the object.

Because the use of space-filling curves for indexing will imply a large number of indices, in contrast a nice possibility is to regroup fractal points into quadrants in order to use quadtrees. The latter are also valuable because they provide the ability to store objects with different sizes. Consequently, geographical objects of large areal extents will be located near the root of the tree and small objects in the terminal leaves.



Peano keys	Side length	Objects
0	4	E
0	2	D
1	1	A
4	1	F
8	2	C
15	1	B,G

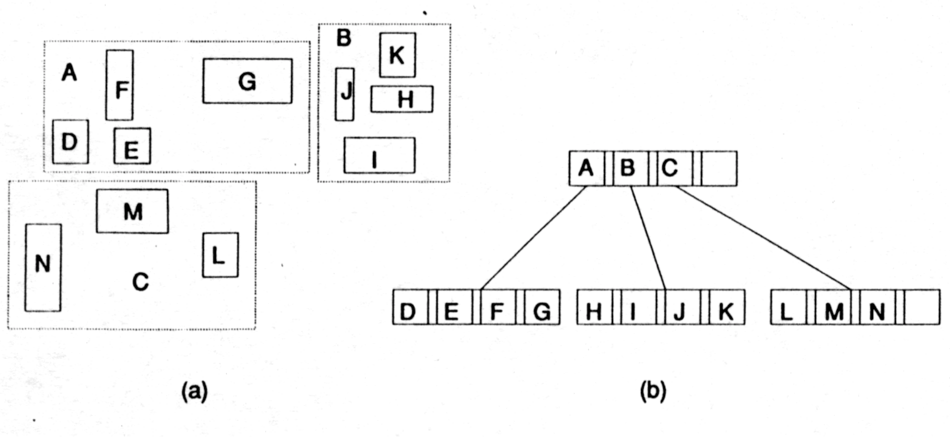
Spatial index with linear quadtree



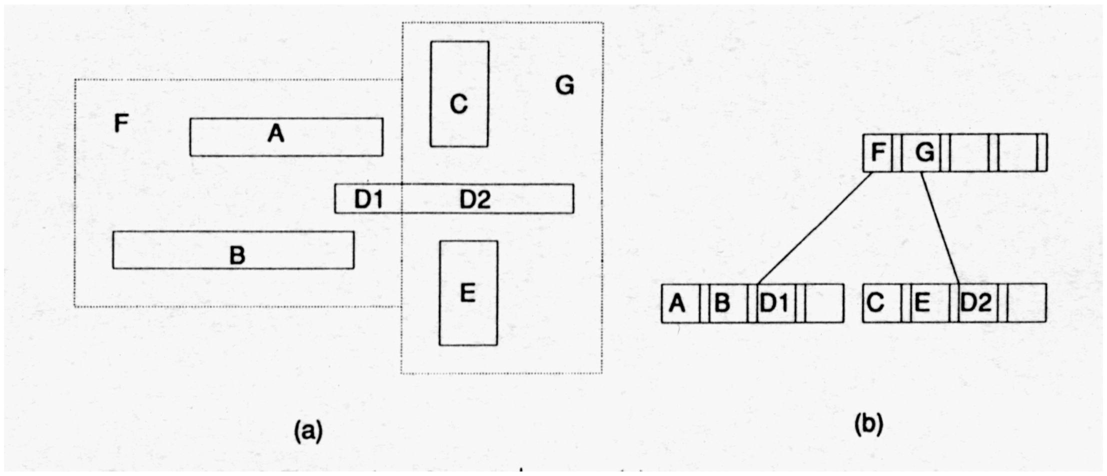
Spatial index with hierarchical quadtree

Indexing by R- and R$^+$-trees

The other possibility for spatial indexing is to use extents bounding spatial objects. One alternative is to use minimum-bounding rectangles, organized either in R-trees or in R$^+$-trees:



R-tree (nonoverlapping rectangles and hierarchical structure)



R+-tree (rectangle split by higher level rectangle and hierarchical structure)

The basic intent behind range trees is to create rectangles aligned with the orthogonal axes of the coordinate spaces, in order to:

1. embrace as many objects as possible; and
2. have as little overlap as possible between rectangles, but
3. allow for subdivision to get smaller boxes within each existing rectangle.

The spatial index is determined as the rectangle in which the object is contained, with a level in a tree conveying information about resolution. Each object is associated with an R-tree node, just as for a quadtree. Precision of location may be determined for coordinate data contained in the relation.

With the relational model, it is very easy to encode an R-tree. In relational forniat, the rectangle description is:

RECT (Rectangle-ID, Type, Min-X, Max-X, Min-Y, Max-Y)

for which Rectangle-ID means any rectangle number so that Min-X, Min-Y, Max-X and Max-Y correspond to the coordinates of its vertices, and Type is the rectangle type, whether real or pseudo. And, for overlappings, pseudo-rectangles can have the same kinds of numbering as the real bounding rectangles.

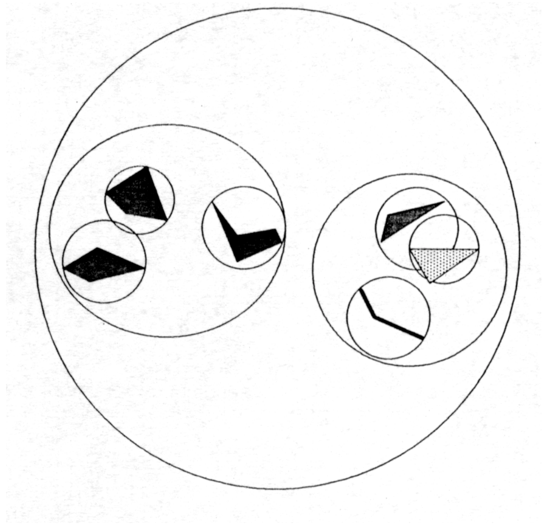
There is also a need for a relation for the assignment of rectangles to higher order units:

PS (Higher-Level-ps-rectangle-ID, Lower-level-ps-rectangle-ID)

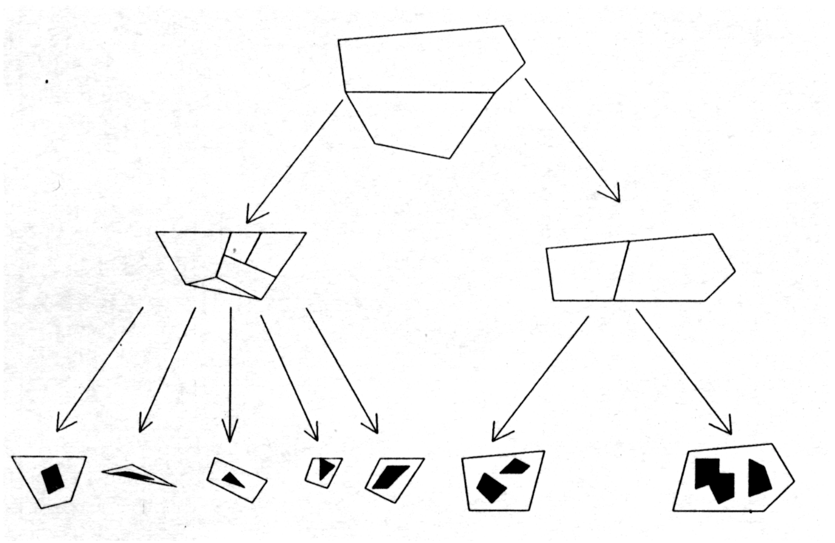
The relation DECOMP (Initial-rectangle-ID, Rectanglel-ID, Rectangle2-ID) can be useful for identifying the assignment of the pieces.

Indexing by other kinds of trees

The main drawbacks of minimum-bounding rectangles is that this way of spatial indexing is very sensitive to orientation. Recently some other methods have been proposed based on spheres and polygons:



Indexing with sphere trees



Indexing with a cell tree

Instead of using a bounding rectangle, Van Oosterom and Classen (1990) have proposed enclosing objects by circles (or spheres at three dimensions). Even though it is often not easy to compute the circle, it is obvious that the extent of this geometric figure is not orientation sensitive.

Moreover, this kind of spatial indexing is insensitive to orientation if the axes are rotated. Perhaps the main challenge is to find a method to determine automatically the bounding circle or sphere for any object; afterwards the addition or deletion of objects is not a problem.

Another possibility is to index using polygons, called ***cell trees*** (Giinther, 1990). In this case, each object is bounded by a convex polygon. The main challenge is to determine rapidly the convex polygon for bounding the objects, especially the number of sides.

Also, Faloutsos and Rong (1989) have combined the R-tree and fractals by a so-called ***double transformation***. Rectangles, defined by minimum and maximum ***x*** and ***y***, can be represented by a point in a four-dimensional space (the min-X, min-Y, max-X and max-Y); this represents the first transformation.

Then, all 4D points representing rectangles are ordered by four-dimensional Hilbert or Peano keys, being the second transformation. Their results show that 4D Hilbert keys give the better performance for their criteria.

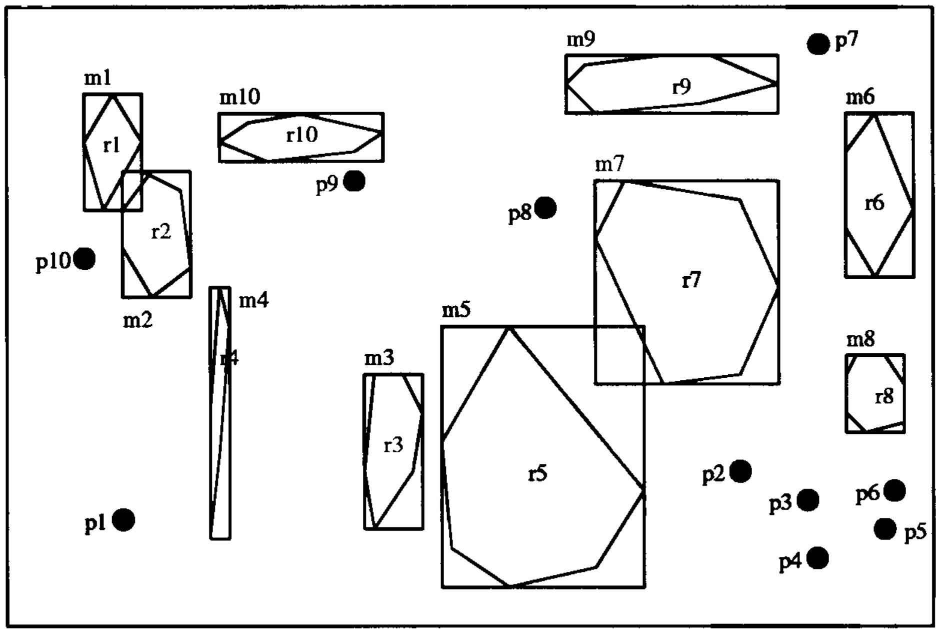
Some practical aspects of spatial indexing

As a practical matter, only a few commercial spatial information systems today provide spatial indexing capabilities. Some systems allow access to database objects via mouse or other graphic cursor input for points or boxes or other shapes. Otherwise there is access via names or numerical identifiers in the attribute data tables. Sometimes topological neighbourhoods provide a means of access, by following line segment or graph links for a specified polygon or line. Indexing capabilities are much rarer. For one commercial system in which indexing tools are made available, the user manuals for the ARC/INFO system (ESRI) indicate that indices for both attributes and the spatial domain can be created. The latter indexing process uses adaptive grid-cells, the former use a binary searching mechanism, operating on data stored as modified binary (B-) trees.

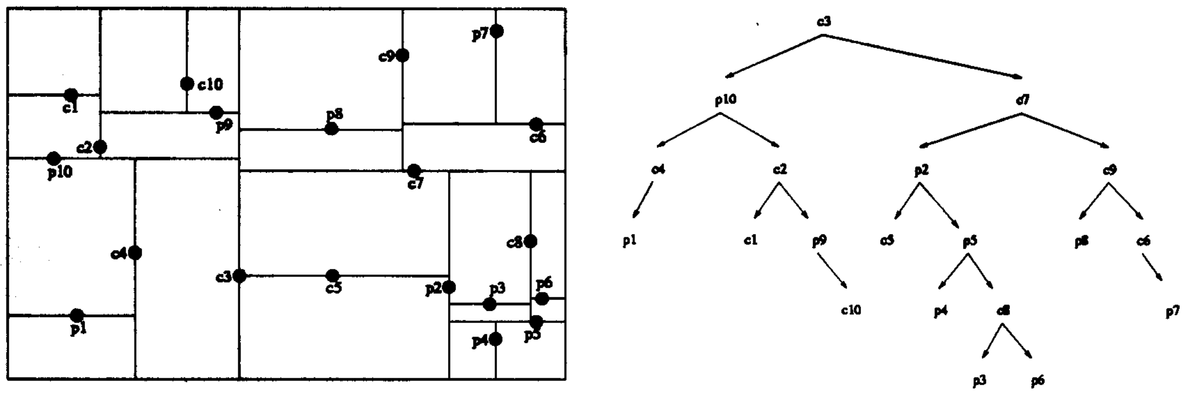
The task of spatial indexing is very challenging. At present there are several techniques but none emerges as the best; although some form of hierarchical organization is generally advantageous. Moreover, two main secondary issues must also be solved: multi-layer indexing, and taking the physical disk structure into account. In several practical situations, spatial databases are split into several layers, each of them concerning a particular theme, for instance, a layer for streets, a layer for gas networks, a layer for sewerage, and so on. For this type of database it is interesting to create as many indices as there are layers and, for practical reasons, different indices may be established for different types of spatial unit. But when it is desirable to work with several thematic layers within one cover area, then the layers must be combined adequately. With a structure such as Peano keys it is simple to merge two indices, but for Rtrees and cell trees, the tree branches must be redetermined, a time consuming task.

Other Multidimensional Access Methods

Running example:

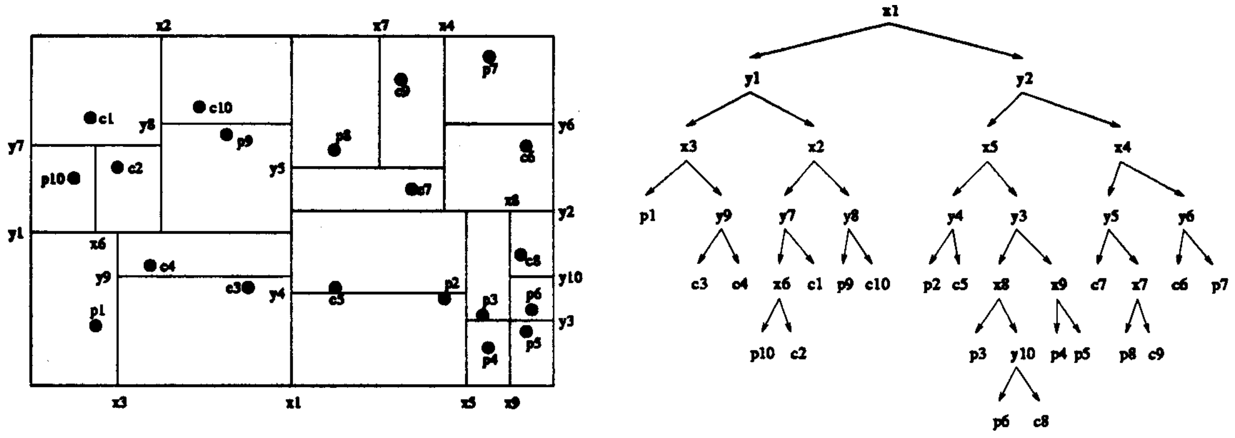


K – D – Tree



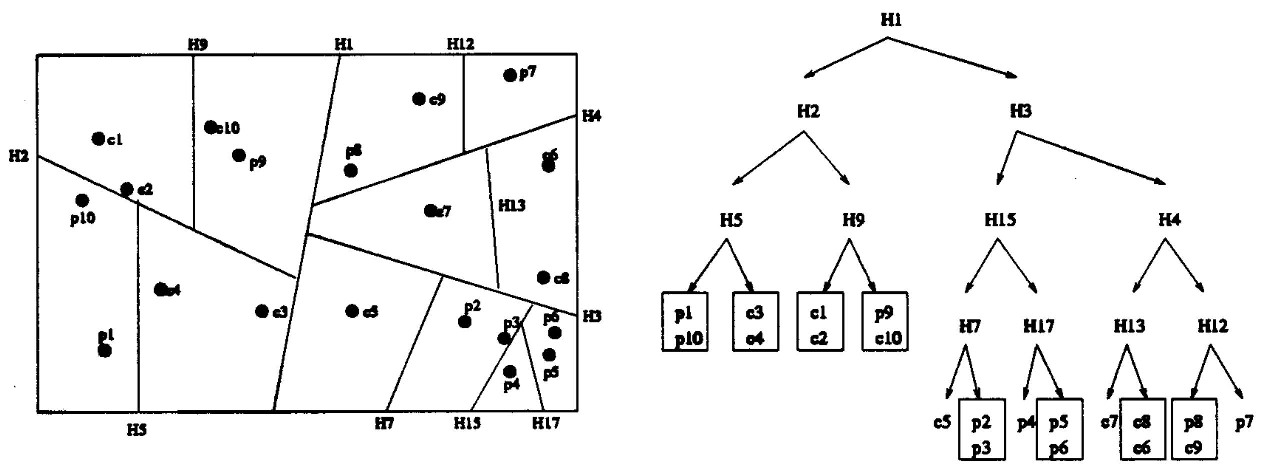
k-d-tree is a binary search tree that represents the recursive subdivision of the universe into subspaces by means of (d-1)-dimensional hyperplanes.

Adaptive k-d-tree



Adaptive k-d-tree choose a split such that one finds about the same number of elements on both sides. While the splitting hyperplanes are still parallel to the axes, they do not have to contain a data point and their directions do not have to be strictly alternating anymore. As a result, the split points are not part of the input data; all data points are stored in the leaves. Interior nodes contain the dimension (e.g. x or y) and the coordinate of the corresponding split. Splitting is continued recursively until each subspace contains only a single point. The adaptive k-d-tree is not a very dynamic structure; it is obviously difficult to keep the tree balanced in the presence of frequent insertions and deletions. The structure works best if all the data is known a priori and if updates are rare.

BSP Tree



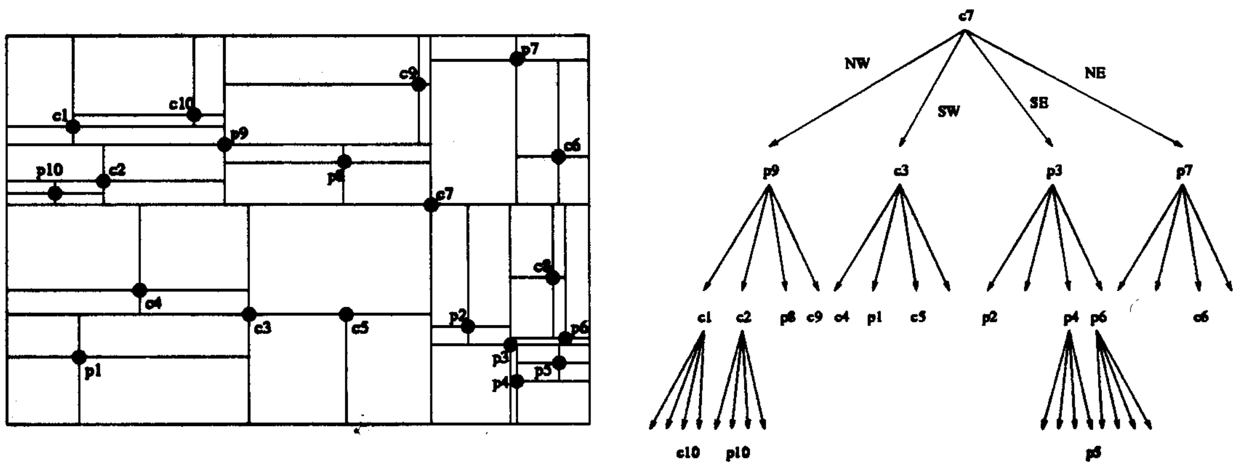
Splitting the universe only along iso-oriented hyperplanes is a severe restriction. Allowing arbitrary orientations gives more flexibility to find a hyperplane that is well-suited for the split. The ***binary space partitioning (BSP)*** tree are binary trees that represent a recursive subdivision of the universe into subspaces by means of (d - 1)- dimensional hyperplanes.

Each subspace is subdivided independently of its history and of the other subspaces. The choice of the partitioning hyperplanes depends on the distribution of the data points in a given subspace. The decomposition usually continues until the number of points in each subspace is below a given threshold.

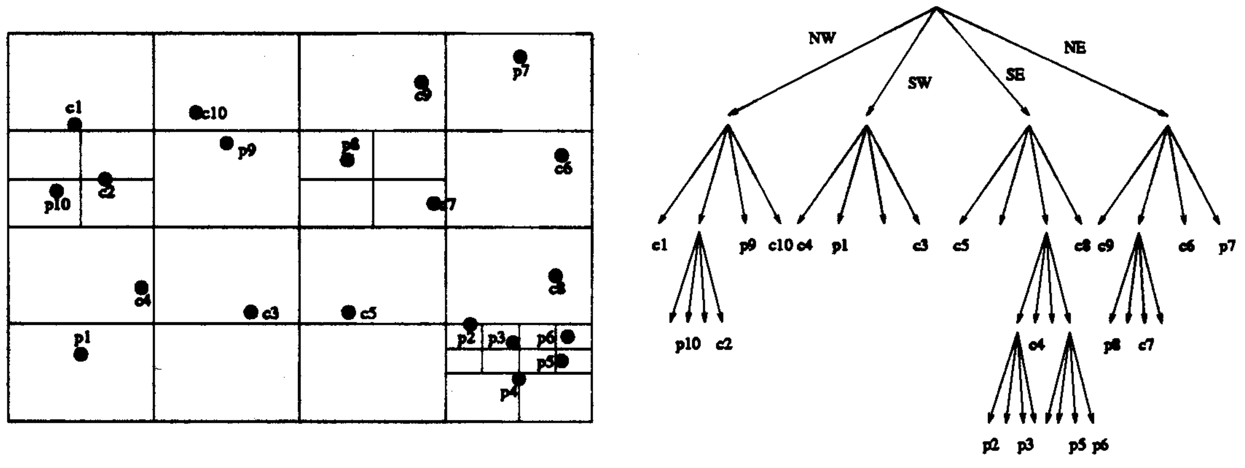
The resulting partition of the universe can be represented by a BSP tree, where each hyperplane corresponds to an interior node of the tree and each subspace corresponds to a leaf. Each leaf stores references to those data points that are contained in the corresponding subspace. Figure shows a BSP tree for the running example with no more than two data points per subspace.

The Quadtree

Point quadtree

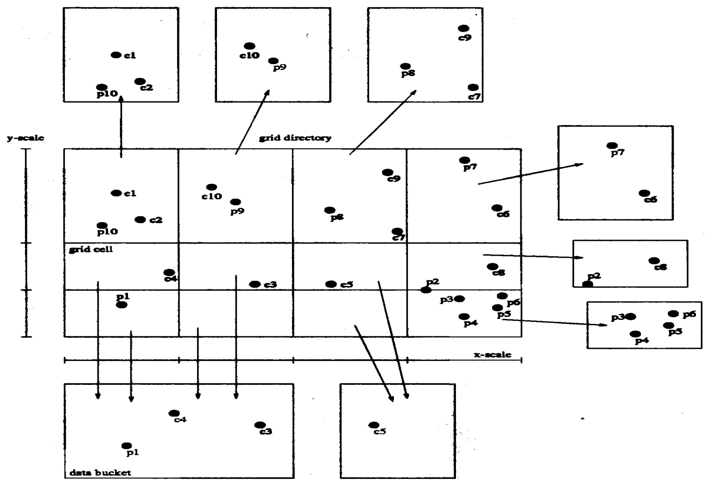


Region tree

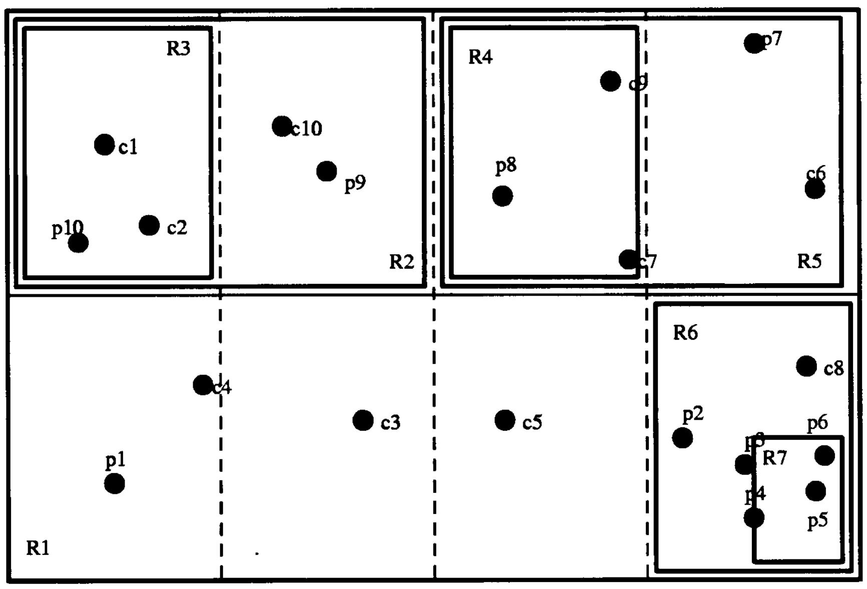


Point Access Methods

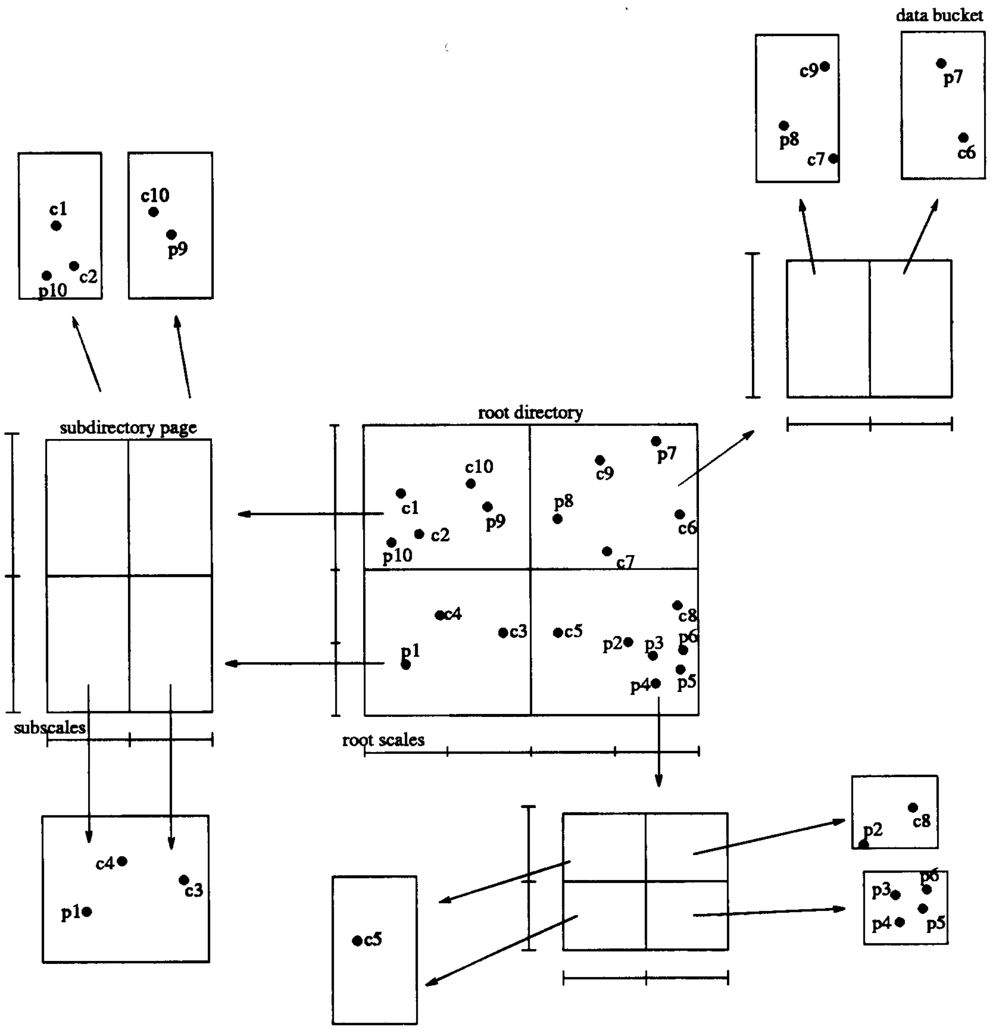
Grid File



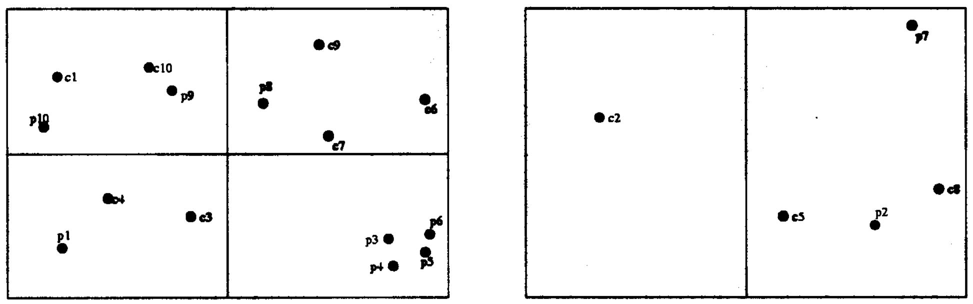
BANG File



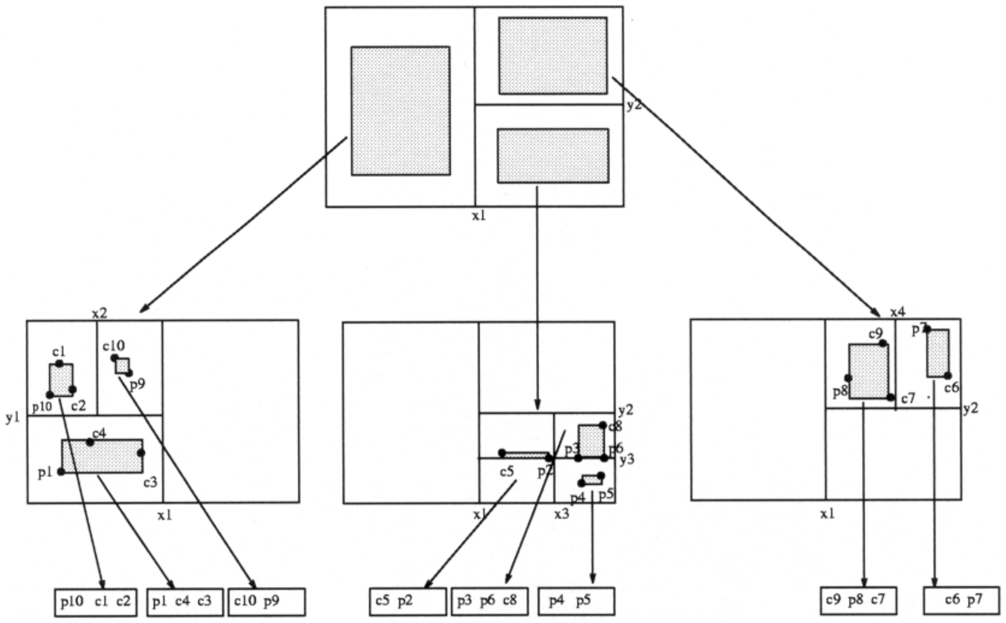
Two-Level Grid File



Twin Grid File



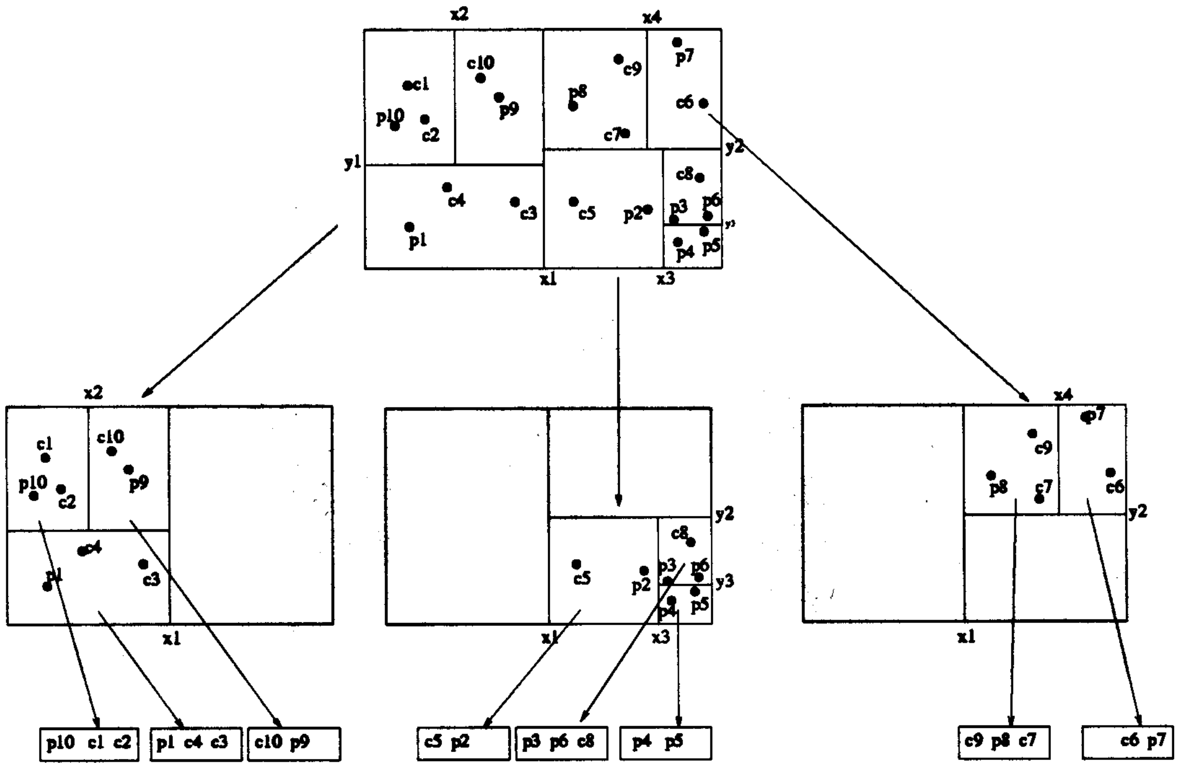
Buddy Tree



The buddy tree is a dynamic hashing scheme with a tree-like directory. The universe is cutted recursively into two parts of equal size with iso-oriented hyperplanes, and each interior node corresponds to a partition together with interval. The interval corresponds to MBB, covering points below of given node. Also:

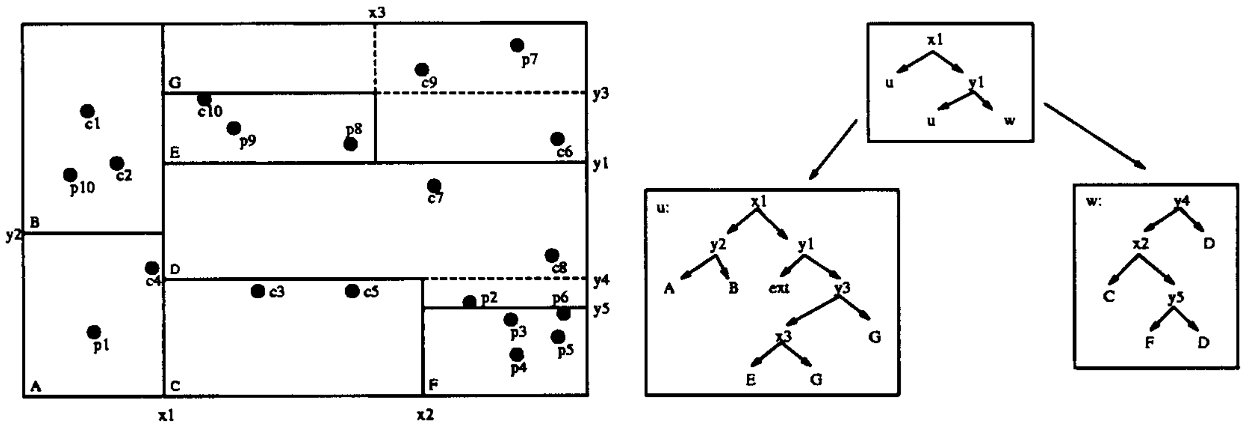
- Each directory node contains at least two entries;
- Whenever a node is split, the MBB and subnodes are recomputed, to fit situation;
- Except for the root of the directory, there is exactly one pointer referring to each directory page.

K-D-B-Tree



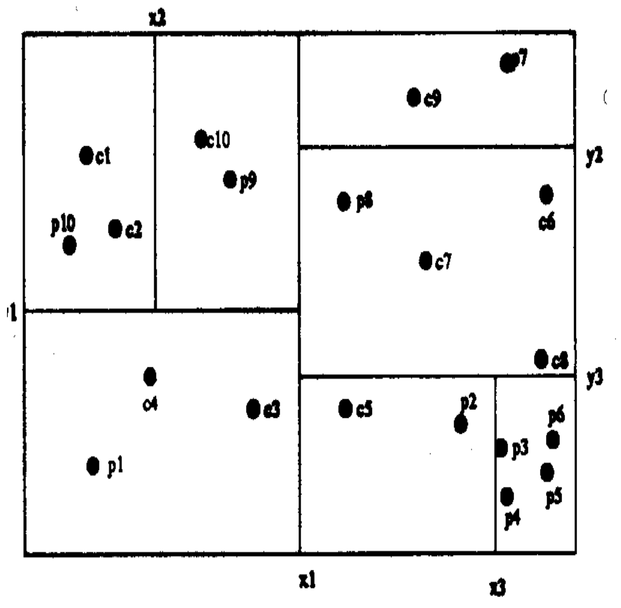
The k-d-B-tree combines properties of the adaptive k-d-tree and the B-tree.

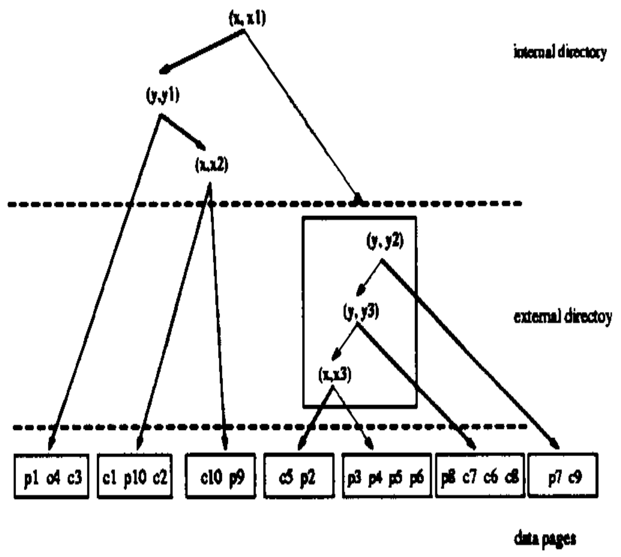
hB-tree



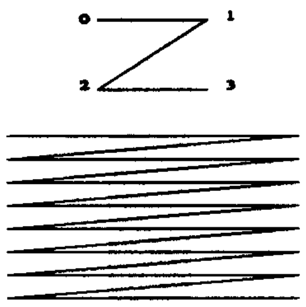
The hB-tree (holey brick tree) is similar to k-d-B-tree, except that splitting of the node is done based on ***multiple attributes***, the result is somewhat fractal structure, with external ***enclosing regions*** and several cavities called ***extracted regions***.

LSD Tree

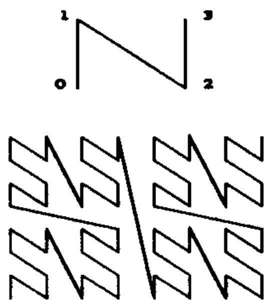




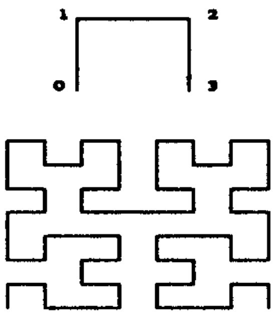
Space-filling curves



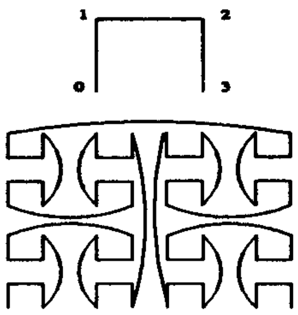
(a)



(b)

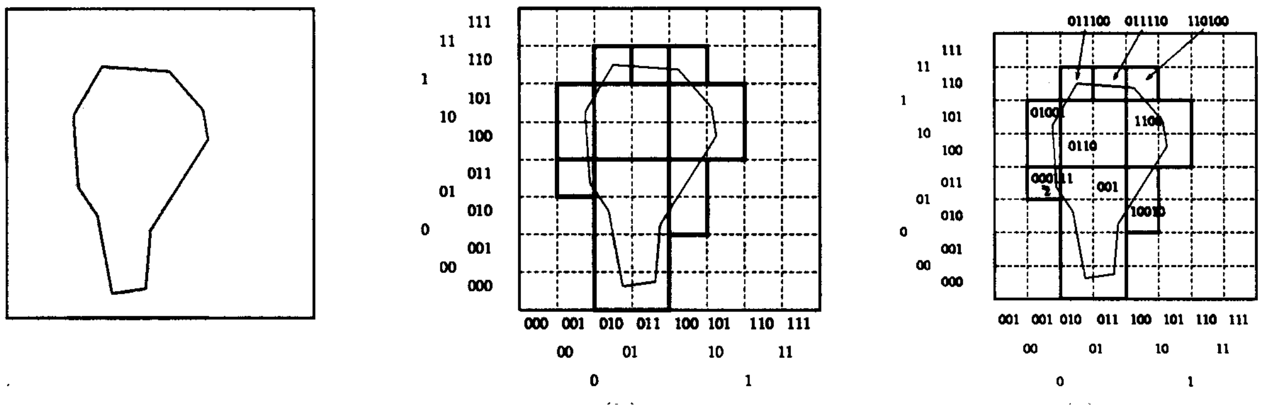


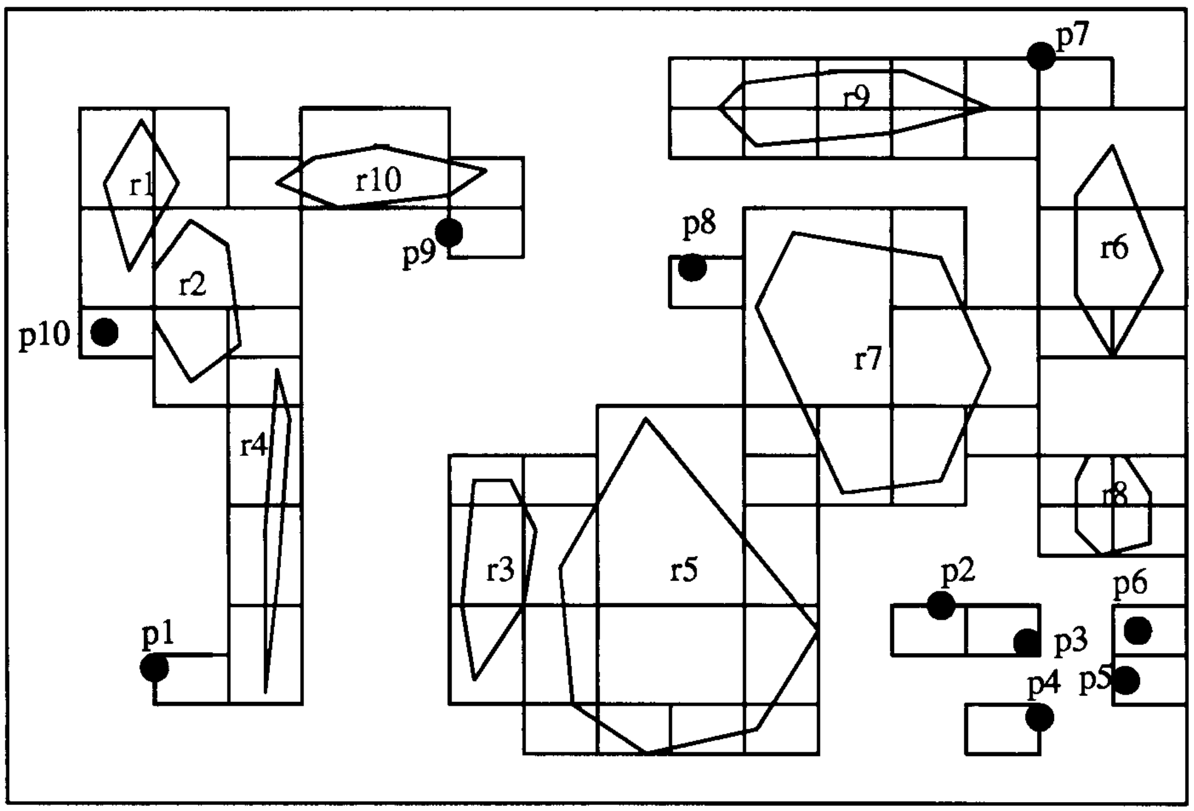
(c)



(d)

Z-ordering



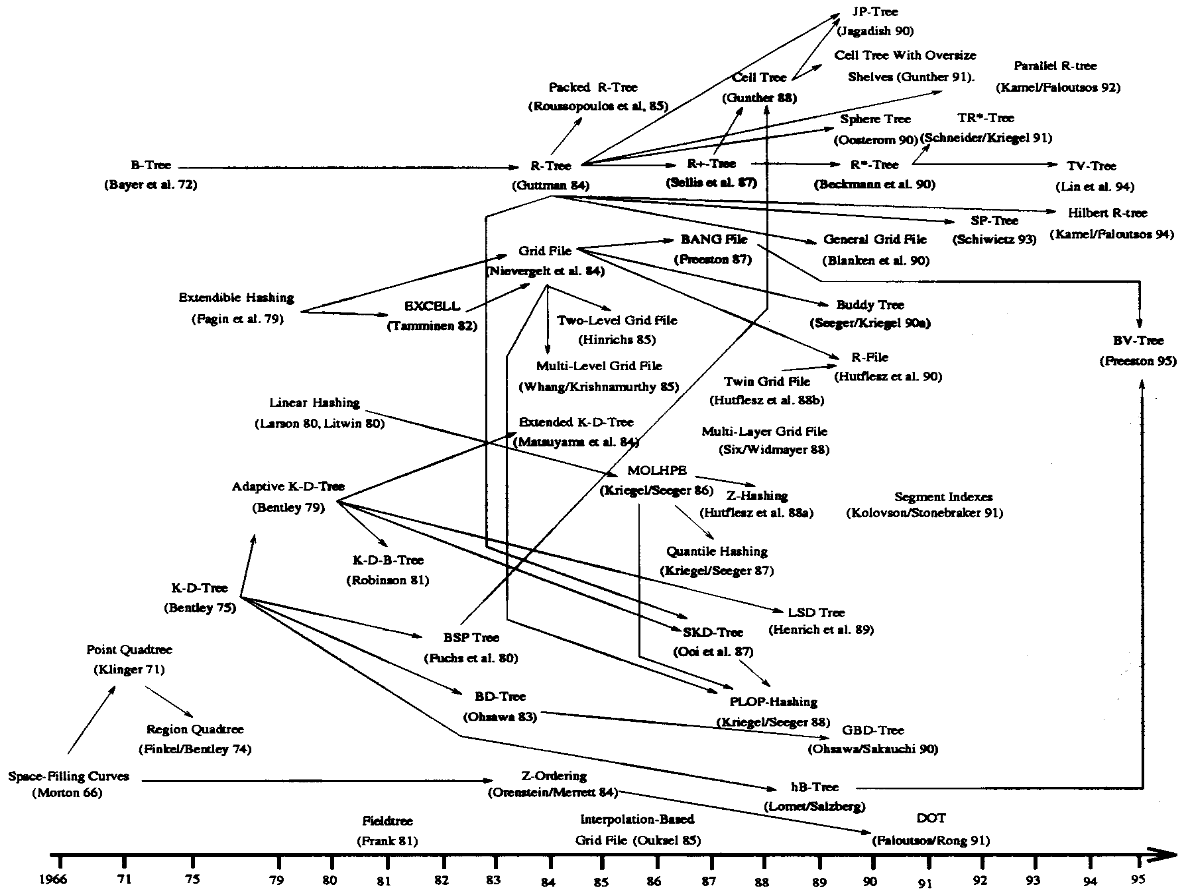


Basic properties of spatial data:

1. spatial data has a complex structure (a spatial data object may be composed of a single point or several thousands of point sets, arbitrarily distributed across space. It is usually not possible to store collections of such objects in a single relational table with a fixed tuple size)
2. spatial data is often dynamic (insertions and deletions are interleaved with updates, and data structures used in this context have to support this dynamic behavior)

3. spatial databases tend to be large (the seamless integration of secondary and tertiary memory is therefore essential for efficient processing)
4. there is no standard algebra defined on spatial data (no standardized set of base operators. The set of operators heavily depends on the given application domain)
5. many spatial operators are not closed (the intersection of two polygons, for example, might return any number of single points, dangling edges, or disjoint polygons)
6. although the computational costs vary between operators, spatial database operators are generally more expensive than standard relational operators

History of Multidimensional Access Methods


JP-Tree (Jagadish 90)
Cell Tree With Oversize Shelves (Gunther 91).
Parallel R-tree (Kamel/Faloutsos 92)
Packed R-Tree (Roussopoulos et al. 85)
Cell Tree (Gunther 88)
Sphere Tree (Oosterom 90)
TR*-Tree (Schneider/Kriegel 91)
B-Tree (Bayer et al. 72)
R-Tree (Guttman 84)
R+-Tree (Sellis et al. 87)
R*-Tree (Beckmann et al. 90)
TV-Tree (Lin et al. 94)
SP-Tree (Schiwietz 93)
Hilbert R-tree (Kamel/Faloutsos 94)
Grid File (Nievergelt et al. 84)
BANG File (Freeston 87)
General Grid File (Blanken et al. 90)
Extendible Hashing (Fagin et al. 79)
EXCELL (Tamminen 82)
Two-Level Grid File (Hinrichs 85)
Buddy Tree (Seeger/Kriegel 90a)
BV-Tree (Freeston 95)
Multi-Level Grid File (Whang/Krishnamurthy 85)
R-File (Hutflesz et al. 90)
Twin Grid File (Hutflesz et al. 88b)
Linear Hashing (Larson 80, Litwin 80)
Extended K-D-Tree (Matsuyama et al. 84)
Multi-Layer Grid File (Six/Widmayer 88)
MOLHPE (Kriegel/Seeger 86)
Z-Hashing (Hutflesz et al. 88a)
Segment Indexes (Kolovson/Stonebraker 91)
Adaptive K-D-Tree (Bentley 79)
Quantile Hashing (Kriegel/Seeger 87)
K-D-B-Tree (Robinson 81)
K-D-Tree (Bentley 75)
LSD Tree (Henrich et al. 89)
SKD-Tree (Ooi et al. 87)
Point Quadtree (Klinger 71)
BSP Tree (Fuchs et al. 80)
BD-Tree (Ohsawa 83)
PLOP-Hashing (Kriegel/Seeger 88)
GBD-Tree (Ohsawa/Sakauchi 90)
Region Quadtree (Finkel/Bentley 74)
Space-Filling Curves (Morton 66)
Z-Ordering (Orenstein/Merrett 84)
hB-Tree (Lomet/Salzberg)
Fieldtree (Frank 81)
Interpolation-Based Grid File (Ouksel 85)
DOT (Faloutsos/Rong 91)
1966
71
75
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95