S

Spatial Indexing Techniques

Yannis Manolopoulos[1], Yannis Theodoridis[2], and Vassilis J. Tsotras[3]
[1]Aristotle University, Thessaloniki, Greece
[2]University of Piraeus, Piraeus, Greece
[3]University of California-Riverside, Riverside, CA, USA

Synonyms

Spatial access methods

Definition

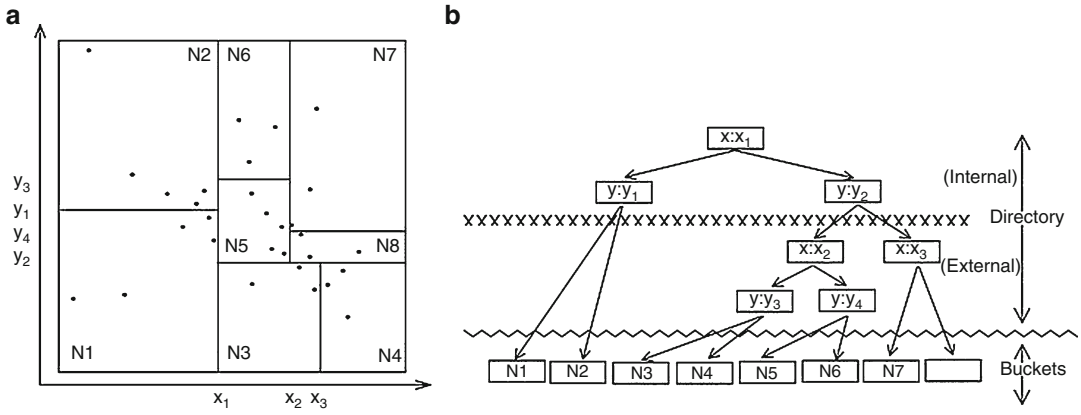
A spatial index is a data structure designed to enable fast access to spatial data. Spatial data come in various forms, the most common being points, lines, and regions in n-dimensional space (practically, $n = 2$ or 3 in geographical information system (GIS) applications). Typical "selection" queries include the *spatial range query* ("find all objects that lie within a given query region") and the *spatial point query* ("find all objects that contain a given query point"). In addition, multidimensional data introduce spatial relationships (such as overlapping and disjointness) and operators (e.g., nearest neighbor), which need to be efficiently supported as well. Example queries are the *spatial join query* ("find all pairs of objects that intersect each other") and the *nearest neighbor query* ("find the five objects nearest to a given query point"). It should be noted that traditional indexing approaches (B^+-trees, hashing, etc.) are not appropriate for indexing spatial data; the basic reason is the lack of total ordering, which is an inherent characteristic in a multidimensional space. As a result, specialized access methods are necessary.

Historical Background

Many applications (VLSI, CAD/CAM, GIS, multimedia) need to represent, store, and manipulate spatial data types, such as points, lines, and regions in n-dimensional space. Although the representation of this type of data may be straightforward in a traditional database system (e.g., a two-dimensional point may be represented as a pair of x- and y- numeric values), spatial relationships (e.g., overlapping) and operators (e.g., nearest neighbor) need to be efficiently supported as well. These spatial relationships and operators have led to a variety of interesting and more complex queries like spatial joins, nearest neighbors, etc. As a result, specialized access methods have been proposed in order to quickly answer the above complex queries, as well as spatial range/point queries.

Given the characteristics of spatial data, for each spatial operator the query object's geometry needs to be combined with each data object's geometry. Nevertheless, the processing of com-

L. Liu, M.T. Özsu (eds.), *Encyclopedia of Database Systems*,
DOI 10.1007/978-1-4899-7993-3_355-3



Spatial Indexing Techniques, Fig. 1 The LSD-tree

plex geometry representations, usually polygons, is very expensive in terms of CPU cost. For that reason, the object geometries are approximated (typically by minimum bounding rectangles (MBRs)), and these approximations are then stored in underlying indices while the actual geometry is stored separately. As a result, a two-step procedure is involved during query processing, consisting of a *filter step* and a *refinement step*. The question that arises is how the object approximations (MBRs) are organized in order to answer the hits and the candidates, i.e., the result of the filter step.

Various spatial indices have been proposed in the literature and can be divided in two categories: indices designed for multidimensional points and indices for multidimensional regions. Examples in the first category are the LSD-tree [7], the grid file [10], the hB-tree [9], the buddy tree [14], and the BV-tree [3]. The major representatives in second category are the R-tree [5] and the quadtree [2] and their variants.

Given the complexity of the indexing problem and the different requirements of the multiple applications that index spatial data, it is not clear which the best index is. Nevertheless, R-tree implementations have found their way into commercial DBMSs. This is mainly due to their simplicity and ease of implementation (their structure is an adaptation of the B^+-tree for spatial data), as well as their robust performance for many applications.

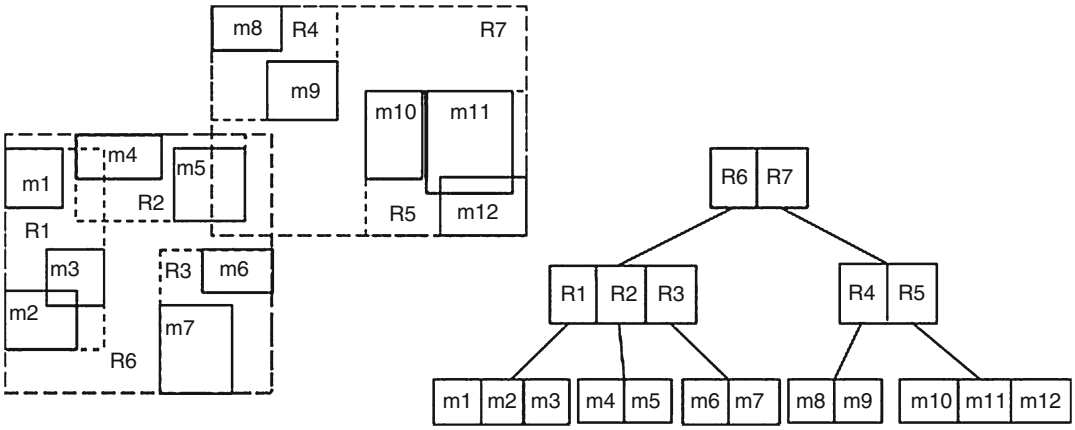
Foundations

This section first discusses indices for multidimensional points, while the description of major indices for multidimensional (non-point) regions follows.

Indices for Multidimensional Points

The LSD-tree (local split decision tree), proposed in [1], maintains a catalog that separates space in a collection of (non-equal sized) disjoint subspaces using the extended k-d tree structure. New entries are inserted into the appropriate bucket. When an overflow happens, then the bucket is split, and the information about the partition line (split dimension and split position) is stored in a directory. Thus the overall structure of the LSD-tree consists of data buckets and a directory tree. The directory tree is kept in main memory until it grows more than a threshold; then a sub-tree is stored in an external catalogue in order for the whole structure to remain balanced (an example appears in Fig. 1). Inserting a new entry (point) in the LSD-tree is straightforward since nodes are disjoint. However, the target node may overflow due to an insertion; a split procedure then takes place.

The LSD-tree is a *space-driven structure*, i.e., it decomposes the complete workspace. Other members of this family include the grid file [10] and the hB-tree [9]. On the other hand, *data-driven structures* only cover those parts of the



Spatial Indexing Techniques, Fig. 2 The R-tree

workspace that contain data objects. Examples are the buddy tree [14] and the BV-tree [3].

The grid file is an access method comprising of two separate parts: (i) the *directory* and (ii) the *linear scales*. The grid file imposes a grid on the indexed multidimensional attribute space. Each cell in this grid corresponds to one data page. The data points that "fall" inside a given cell are stored in the cell's corresponding page. Each cell must thus store a pointer to its corresponding page. This information is stored in the grid file's *directory*. The information of how each dimension is divided (and thus how data values are assigned to cells) is kept in the *linear scales*. The grid file can be thought as a multidimensional extension of hashing. As a result, exact match queries take only two disk accesses, one for the directory and one for the data page.

Indices for Multidimensional Regions

As with point indexing, two different approaches (data driven and space driven) have been proposed for indexing regions as well. The main representatives are the R-tree [5] and the quadtree, [2, 13], which were later followed by dozens of variants. In the sequel, the two structures are presented in detail. The reader is referred to a recent exhaustive survey [4] for further reading on their variants.

R-trees were originally proposed [5] as a direct extension of B^+-trees in n-dimensional space. The data structure is a height-balanced tree that consists of intermediate and leaf nodes. A leaf node is a collection of entries of the form (o_id, R) where o_id is an object identifier, used to refer to an object in the database, and R is the minimum bounding rectangle (MBR) approximation of the data object. An intermediate node is a collection of entries of the form (ptr, R) where ptr is a pointer to a lower level node of the tree and R is a representation of the minimum rectangle that encloses all MBRs of the lower-level node entries. Let M be the maximum number of entries in a node and let $m \leq M/2$ be a parameter specifying the minimum number of entries in a node. An R-tree satisfies the following properties: (i) every leaf node contains between m and M entries unless it is the root; (ii) for each entry (o_id, R) in a leaf node, R is the MBR minimum bounding rectangle approximation of the object represented by o_id; (iii) every intermediate node has between m and M children unless it is the root; (iv) for each entry (ptr, R) in an intermediate node, R is the smallest rectangle that completely encloses the rectangles in the child node; (v) the root node has at least two children unless it is a leaf; and (vi) all leaves appear at the same level. As an example, Fig. 2 illustrates several minimum bounding rectangles (MBRs) m_i and the corresponding R-tree built on these rectangles (assuming maximum node capacity $M = 3$).

In order for a new entry E to be inserted into the R-tree, starting from the root node, the child that needs minimum enlargement to include E

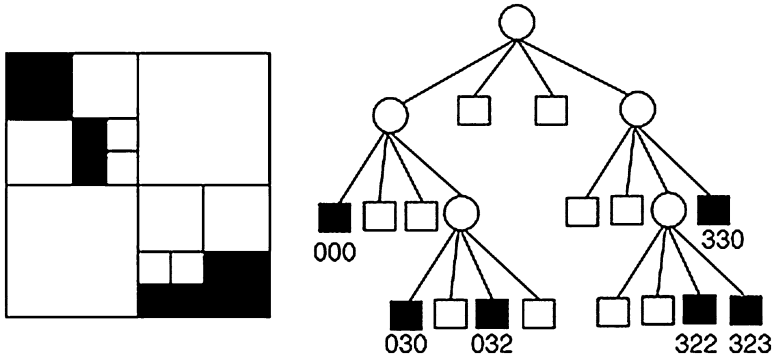
is chosen (ties are resolved by choosing the one with the smallest area). When a leaf node N is reached, E is inserted into that, probably causing a split if N is already full. In such a case, the existing entries together with E are redistributed in two nodes (the current and a new one) with respect to the minimum enlargement criterion. In the original paper [6], three alternatives were proposed in order to find the two groups: an exhaustive, a quadratic-cost, and a linear-cost split algorithm. The processing of a point or range query with respect to a query window q (which could be either point or rectangle, respectively) is straightforward: starting from the root node, several tree nodes are traversed down to the leaves, depending on the result of the overlap operation between q and the corresponding node rectangles. When the search algorithm reaches the leaf nodes, all data rectangles that overlap the query window q are added to the answer set. Regarding k-nearest-neighbor queries, [12] proposed customized branch-and-bound algorithms for R-trees.

After Guttman's proposal, several researchers proposed their own improvements on the basic idea. Roussopoulos and Leifker [11] proposed the *packed* R-tree for bulk loading data in an R-tree. Objects are first sorted in some desirable order (according to the low-x value, low-y value, etc.) and then the R-tree is bulk loaded from the sorted file and R-tree nodes are packed to capacity. Note that the above techniques allow node "overlapping": MBRs of different nodes can overlap. Since no disjointness is guaranteed, during a search multiple paths of the R-tree may be traversed. An efficient variation, namely, the R^+-tree, was proposed by Sellis et al. [15]. To preserve disjointness among node rectangles, the R^+-tree uses a "clipping" technique that duplicates data entries when necessary. However, the penalty is an (possibly high) increase in space demand due to the replication of data, which, in turn, degenerates search performance. Generally speaking, clipping techniques are ideal for point queries because a single path should be traversed, while range queries tend to be expensive, when compared with the overlapping techniques.

Later, Beckman et al. [1] and Kamel and Faloutsos [8] proposed two R-tree-based methods, the R*-tree and the Hilbert R-tree, respectively, which are currently considered to be the most efficient members of the R-tree family in terms of query performance. The R*-tree uses a rather complex but more effective grouping algorithm to split nodes by computing appropriate area, perimeter, and overlap values, while the Hilbert R-tree actually stores Hilbert values at the leaf level and ranges of those values at the upper levels, similarly to the B^+-tree construction algorithm. In addition, a "lazy" split technique is followed, where overflow entries are evenly distributed among sibling nodes, and only when all those are full, a new node (hence, split) is created.

The *region quadtree* [2] is probably the most popular member in the quadtree family. It is used for the representation of binary images, that is, $2^n \times 2^n$ binary arrays (for a positive integer n), where a "1" ("0") entry stands for a black (white) picture element. More precisely, it is a degree four tree with height n, at most. Each node corresponds to a square array of pixels (the root corresponds to the whole image). If all of them have the same color (black or white), the node is a leaf of that color. Otherwise, the node is colored gray and has four children. Each of these children corresponds to one of the four square sub-arrays to which the array of that node is partitioned. It is assumed here that the first (leftmost) child corresponds to the upper left sub-array, the second to the upper right sub-array, the third to the lower left sub-array, and the fourth (rightmost) child to the lower right sub-array, denoting the directions NW, NE, SW, and SE single ended, respectively. Figure 3 illustrates a quadtree for an 8×8 pixel array. Note that black (white) squares represent black (white) leaves, whereas circles represent internal nodes (also, *gray* ones).

Region quadtrees, as presented above, can be implemented as main memory tree structures (each node being represented as a record that points to its children). Variations of region quadtrees have been developed for secondary memory. *Linear region quadtrees* [8] are the ones used most extensively. A linear quadtree



Spatial Indexing Techniques, Fig. 3 The quadtree

representation consists of a list of values where there is one value for each black node of the pointer-based quadtree. The value of a node is an address describing the position and size of the corresponding block in the image. These addresses can be stored in an efficient structure for secondary memory (such as a B^+-tree). There are also variations of this representation where white nodes are stored too or variations which are suitable for multicolor images. Evidently, this representation is very space efficient, although it is not suited to many useful algorithms that are designed for pointer-based quadtrees. The region quadtree [2] is probably the most popular linear implementations are the fixed length (FL), the fixed depth (FD), and the variable length (VL) linear implementations [13]. Techniques for computing various kinds of geometric properties have also been developed. Connected component labeling, polygon coloring, and computation of various types of perimeters fall in this category. Finally, many operations on images have been developed, for example, point location, set operations on two or more images (intersection, union, difference, etc.), window clipping, linear image transformations, and region expansion.

Other region quadtree variants have appeared in the literature mainly for indexing non-regional data. MX quadtrees are used for storing points seen as black pixels in a region quadtree. PR quadtrees are also used for points. However, points are drawn from a continuous space, in this case. MX-CIF quadtrees are used for small rectangles. Each rectangle is associated with the quadtree node corresponding to the smallest block that contains the rectangle. PMR quadtrees are used for line segments. Each segment is stored in the nodes that correspond to blocks intersected by the segment. A detailed presentation of these and other region quadtree variants is given in [8].

Key Applications

Geographic information systems (GIS) deal extensively with the management of two- and three-dimensional spatial data. For example, a map typically contains point objects (locations of interest), line objects (road segments, highways, rivers, etc.), as well as region objects (lakes, forests, etc.). GIS use spatial indexing as a means to provide fast access to large amounts of spatial data.

Multimedia systems manage multimedia objects like images, text, audio, video, etc. A typical query in such systems is the *similarity* query (i.e., find objects that are similar to a query object according to some measure). To answer these queries, each multimedia object is abstracted by a set of multidimensional points (features). These multidimensional points are then indexed by a spatial index. Similarly, the query object is represented by a multidimensional point. The similarity query is then answered as a nearest neighbor query (i.e., find the nearest neighbor(s) to the point that represents the query).

The World Wide Web has also provided new applications for geographic-related queries and thus spatial indexing. Users can now find maps, driving directions, etc. through specialized web

sites that typically offer the ability to perform spatial queries.

Location-based services provide querying capabilities based on the location of the user (e.g., "find the cheapest gas station within 5 miles of my car"). As the user moves in space, the results of the queries change. Such queries typically have a spatial component, and spatial (and spatiotemporal) indexes are used to provide fast response.

CAD systems use spatial objects to store surfaces and bodies of design objects (for e.g., the wings or the wheels of an airplane). Typical spatial queries involve the proximity of spatial objects, their overlap, etc. Related queries (but mainly in the two-dimensional space) are also relevant for VLSI design systems; here the layout of a chip involves various rectangular regions and overlap and proximity queries are of importance.

Computer games also involve many spatial searches. In such an environment, players move around in a three-dimensional space and need to be able to see parts of (partially hidden) objects, various triggers are initiated if a player passed over them, or an explosion needs to identify the nearby objects that are affected. Spatial indexing is used to improve such query response.

Medical imaging also involves large amounts of two- and three-dimensional spatial data. Consider, for example, X-rays or magnetic resonance imaging (MRI) brain scans. Again, proximity, overlap, and related spatial queries are of interest.

Experimental Results

In general, for every presented method, there is an accompanying experimental evaluation in the corresponding reference.

Datasets

A large collection of real spatial datasets, commonly used for experiments, can be found at *R-tree portal* (URL: http://www.rtreeportal.org/).

URL to Code

R-tree portal (see above) contains the code for most common spatial and spatiotemporal indexes, as well as data generators and several useful links for researchers and practitioners in spatiotemporal databases. Similarly, the spatial index library [6] provides a general framework for developing various spatial indices (URL: http://dblab.cs.ucr.edu/spatialindexlib).

Cross-References

- B-Tree
- GIS
- Grid File (and family)
- Nearest Neighbor Query
- Quadtrees (and Family)
- R-Tree (and Family)
- Spatial Join

Recommended Reading

1. Beckmann N, Kriegel H-P, Schneider R, Seeger B. The R*-tree: an efficient and robust access method for points and rectangles. In: Proceedings of ACM SIGMOD International Conference on Management of Data; 1990, p. 322–31.
2. Finkel RA, Bentley JL. Quad Trees: a data structure for retrieval on composite keys. Acta Informatica. 1974;4(1):1–9.
3. Freeston MA. General solution of the n-dimensional B-tree problem. In: Proceedings of ACM SIGMOD International Conference on Management of Data; 1995. p. 80–91.
4. Gaede V, Guenther O. Multidimensional access methods. ACM Comput Surv. 1998;30(2):170–231.
5. Guttman A. R-trees: a dynamic index structure for spatial searching. In: Proceedings of ACM SIGMOD International Conference on Management of Data; 1984. p. 47–57.
6. Hadjieleftheriou M, Hoel E, Tsotras VJ. SaIL: a spatial index library for efficient application integration. GeoInformatica. 2005;9(4):367–89.
7. Henrich A, Six H-W, Widmayer P. The LSD tree: spatial access to multidimensional point and non point objects. In: Proceedings of 15th International Conference on Very Large Data Bases; 1989. p. 43–53.
8. Kamel I, Faloutsos C. Hilbert R-tree: an improved R-tree using fractals. In: Proceedings of 20th

International Conference on Very Large Data Bases; 1994. p. 500–09.
9. Lomet DB, Salzberg B. The hB-tree: a multiattribute indexing method with good guaranteed performance. ACM Trans Database Syst. 1990;15(4):625–58.
10. Nievergelt J, Hinterberger H, Sevcik KC. The grid file: an adaptable symmetric multikey file structure. ACM Trans Database Syst. 1984;9(1):38–71.
11. Roussopoulos N, Leifker D. Direct spatial search on pictorial databases using packed R-trees. In: Proceedings of ACM SIGMOD International Conference on Management of Data; 1985. p. 17–31.
12. Roussopoulos N, Kelley S, Vincent F. Nearest neighbor queries. In: Proceedings of ACM SIGMOD International Conference on Management of Data; 1995. p. 71–9.
13. Samet H. The design and analysis of spatial data structures. Addison-Wesley; Reading, MA, 1990.
14. Seeger B, Kriegel H-P. The Buddy-tree: an efficient and robust access method for spatial database systems. In: Proceedings of 16th International Conference on Very Large Data Bases; 1990. p. 590–601.
15. Sellis T, Roussopoulos N, Faloutsos C. The R^+-tree: a dynamic index for multidimensional objects. In: Proceedings of 13th International Conference on Very Large Data Bases; 1987. p. 507–18.