International Journal of Shape Modeling
Vol. 14, No. 1 (2008) 15–37

World Scientific
www.worldscientific.com

A Sorting Approach to Indexing Spatial Data*

HANAN SAMET

Center for Automation Research
Institute for Advanced Computer Studies
Computer Science Department
University of Maryland
College Park, Maryland 20742, USA
hjs@cs.umd.edu
http://www.cs.umd.edu/~hjs

Spatial data is distinguished from conventional data by having extent. Therefore, spatial queries involve both the objects and the space that they occupy. The handling of queries that involve spatial data is facilitated by building an index on the data. The traditional role of the index is to sort the data, which means that it orders the data. However, since generally no ordering exists in dimensions greater than 1 without a transformation of the data to one dimension, the role of the sort process is one of differentiating between the data and what is usually done is to sort the spatial objects with respect to the space that they occupy. The resulting ordering is usually implicit rather than explicit so that the data need not be resorted (i.e., the index need not be rebuilt) when the queries change (e.g., the query reference objects). The index is said to order the space and the characteristics of such indexes are explored further.

Keywords: Spatial indexing; Sorting; Geometric data structures.

1991 Mathematics Subject Classification: 22E46, 53C35, 57S20

1. Introduction

The representation of multidimensional data is an important issue in solid modeling as well as in many other diverse fields including computer-aided design (CAD), computational geometry, finite-element analysis, and computer graphics (e.g., [73,74,76]). The main motivation in choosing an appropriate representation is to facilitate operations such as search. This means that the representation involves sorting the data

*This work was supported in part by the National Science Foundation under Grants EIA-00-91474, CCF-05-15241, IIS-07-13501, and IIS-0812377, Microsoft Research, NVIDIA, and the University of Maryland General Research Board.

in some manner to make it more accessible. In fact, the term *access structure* or *index* is often used as an alternative to the term *data structure* in order to emphasize the importance of the connection to sorting.

The most common definition of "multidimensional data" is a collection of points in a higher dimensional space (i.e., greater than 1). These points can represent locations and objects in space as well as more general records where each attribute (i.e., field) corresponds to a dimension and only some, or even none, of the attributes are locational. As an example of nonlocational point data, consider an employee record that has attributes corresponding to the employee's name, address, gender, age, height, weight, and social security number (i.e., identity number). Such records arise in database management systems and can be treated as points in, for this example, a seven-dimensional space (i.e., there is one dimension for each attribute), although the different dimensions have different type units (i.e., name and address are strings of characters; gender is binary; while age, height, weight, and social security number are numbers some of which have are associated with different units). Note that the address attribute could also be interpreted in a locational sense using positioning coordinates such as latitude and longitude readings although the stringlike symbolic representation is far more common.

When multidimensional data corresponds to locational data, we have the additional property that all of the attributes usually have the same unit (possibly with the aid of scaling transformations), which is distance in space. In this case, we can combine the distance-denominated attributes and pose queries that involve proximity. For example, we may wish to find the closest city to Chicago within the two-dimensional space from which the locations of the cities are drawn. Another query seeks to find all cities within 50 miles of Chicago. In contrast, such queries are not very meaningful when the attributes do not have the same type. Nevertheless, other queries such as range queries that seek, for example, all individuals born between 1940 and 1960 whose weight ranges between 150 and 200 pounds are quite common and can be posed regardless of the nature of the attributes.

When the range of multidimensional data spans a continuous physical space (i.e., an infinite collection of locations), the issues become more interesting. In particular, we are no longer just interested in the locations of objects, but, in addition, we are also interested in the space that they occupy (i.e., their extent). Some example objects with extent include line segments (e.g., roads, rivers), intervals (which can correspond to time as well as space), regions of varying shape and dimensionality (e.g., lakes, counties, buildings, crop maps, polygons, polyhedra), and surfaces. The objects (when they are not points) may be disjoint or could even overlap.

The fact that the objects have extent has a direct effect on the type of indexes that we need. This can be best understood by examining the nature of the queries that we wish to support. For example, consider a database of objects. There are three types of queries that can be posed to such a database [35]. The first is the set of queries about the objects themselves such as finding all objects that contain a given point or set of points, have a non-empty intersection with a given object,

have a partial boundary in common, have a boundary in common, have any points in common, contain a given object, included in a given object, etc. The second consists of proximity queries such as the nearest object to a given point or object, and all objects within a given distance of a point or object (also known as a range or window query). The third consists of queries involving non-spatial attributes of objects such as given a point or object, finding the nearest object of a particular type, the minimum enclosing object of a particular type, or all the objects of a particular type whose boundary passes through it.

Being able to support the different types of queries described above has a direct effect on the type of indexes that are useful for such data. In particular, recall our earlier observation that a record in a conventional database may be considered as a point in a multidimensional space. For example, a straight line segment object having endpoints (x_1, y_1) and (x_2, y_2) can be transformed (i.e., represented) as the point (x_1, y_1, x_2, y_2) in a 4-d space (termed a *corner transformation* [83])[a]. This representation is good for queries about the line segments (the first type), while it is not good for proximity queries (i.e., the second and third type) since points outside the object are not mapped into the higher dimensional space. In particular, the representative points of two objects that are physically close to each other in the original space (e.g., 2-d for line segments) may be very far from each other in the higher dimensional space (e.g., 4-d), thereby leading to large search regions. This is especially true if there is a great difference in the relative size of the two objects (e.g., a short line segment in proximity to a long line segment as in Figure 1). On the other hand, when the objects are small (e.g., their extent is small), then the method works reasonably well as the objects are basically point objects. The problem is that the transformation only transforms the space occupied by the objects and not the rest of the space (e.g., the query point). Proponents of the transformation method argue that this problem can be overcome by projecting back to original space and indexing on the projection (e.g., [90]). However, at this point, it is not unreasonable to ask why we bother to make the transformation in the first place.

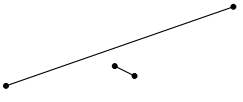


Figure 1: Example of two objects that are close to each other in the original space but are not clustered in the same region of the transformed space when using a transformation such as the corner transformation.

It is important to observe that our notion of *sorting* spatial objects is more

[a]Although for ease of visualization, our discussion and examples are in terms of line segment and rectangle objects, it is applicable to data of arbitrary dimension such as polyhedra and hyperrectangles.

one of differentiating between the objects which is different from the conventional one which is intimately tied to the notion of providing an ordering. As we know, such an ordering implies a linearization which restricts the underlying data to one dimension, and such an ordering usually does not exist in dimensions d higher than one save for a dominance relationship (e.g., [55]) where point $a = \{a_i | 1 \leq i \leq d\}$ is said to dominate point $b = \{b_i | 1 \leq i \leq d\}$ if $b_i \leq a_i, 1 \leq i \leq d$. On the other hand, it is clear that the rationale for our discussion is that the data in which we are interested is of dimension greater than one. This leads to the conclusion that what is needed is an index that sorts (i.e., differentiates) between objects on the basis of spatial occupancy (i.e., their spatial extent). In other words, it sorts the objects relative to the space that they occupy, and this is the focus of the rest of this paper.

Before choosing a particular index we should also make sure that the following requirements are satisfied. First of all, the index should be compatible with the type of data (i.e., spatial objects) that is being stored. In other words, it should enable users to distinguish between different objects as well as render the search efficient in terms of pruning irrelevant objects from further consideration. Second, we must have an appropriate zero or reference point. In the case of spatial occupancy, this is usually some easily identified point or object (e.g., the origin of the multidimensional space from which the objects are drawn). Most importantly, given our observation about the absence of an ordering, it is best to have an implicit rather than an explicit index.

In particular, an implicit index is needed because it is impossible to foresee all possible queries in advance. For example, in the case of spatial relationships such as left, right, up, down, etc. it is impractical to have a data structure which has an attribute for every possible spatial relationship. In other words, the index should support the ability to derive the spatial relationships between the objects. It should be clear that an implicit index is superior to an explicit index, which, for example in the case of two-dimensional data such as the locations of cities, sorts the cities on the basis of their distance from a given point. The problem is that this sorting order is inapplicable to other reference points. In other words, having sorted all of the cities in the US with respect to their distance from Chicago, the result is useless if we want to find the closest city to New Orleans that satisfies a particular condition like having a population greater than 50,000 inhabitants. Therefore, having an implicit index means that we don't have to resort the data for queries other than updates.

2. Methods Based on Spatial Occupancy

The indexing methods that are based on sorting the spatial objects by spatial occupancy essentially decompose the underlying space from which the data is drawn into regions called *buckets* in the spirit of classical hashing methods, with the difference that the spatial indexing methods preserve order. In other words, objects

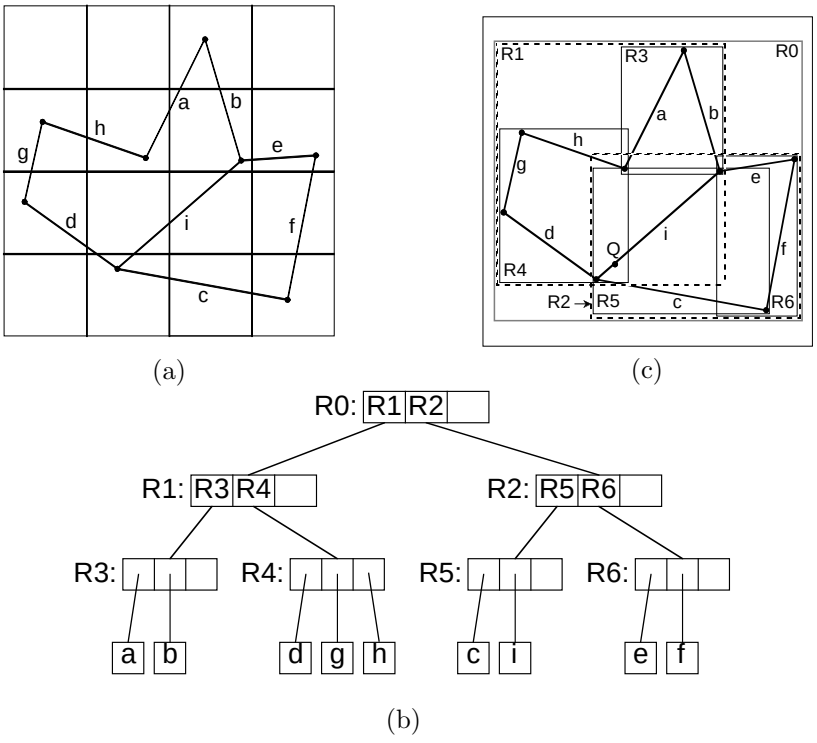


Figure 2: (a) Example collection of straight line segments embedded in a 4×4 grid, (b) the object hierarchy for the R-tree corresponding to the objects in (a), and (c) the spatial extent of the minimum bounding rectangles corresponding to the object hierarchy in (b). Notice that the leaf nodes in the (c) also store bounding rectangles although this is only shown for the nonleaf nodes.

in close proximity should be placed in the same bucket or at least in buckets that are close to each other in the sense of the order in which they would be accessed (i.e., retrieved from secondary storage in case of a false hit, etc.).

There are two principal methods of representing spatial data. The first is to use an object hierarchy that initially aggregates objects into groups, preferably based on their spatial proximity, and then uses proximity to further aggregate the groups thereby forming a hierarchy, where the number of objects that are aggregated in each node of the hierarchy is permitted to range between parameters $m \leq \lceil M/2 \rceil$ and M. The rationale for choosing this type of a range is for the hierarchy to mimic the behavior of a B-tree (e.g., [19]), where each element of the hierarchy acts like a disk page and thus is guaranteed to be half full, provided that $m = \lceil M/2 \rceil$.

Note that the object hierarchy is not unique as it depends on the manner in which the objects were aggregated to form the hierarchy (e.g., minimizing overlap between objects or coverage of the underlying space). Queries are facilitated by also associating a minimum bounding box with each object and group of objects as this

enables a quick way to test if a point can possibly lie within the area spanned by the object or group of objects. A negative answer means that no further processing is required for the object or group while a positive answer means that further tests must be performed. Thus the minimum bounding box serves to avoid wasting work. Equivalently, it serves to differentiate (i.e., "sort") between occupied and unoccupied space. Data structures that make use of axis-aligned bounding boxes (AABB) such as the R-tree [31] and the R*-tree [12] illustrate the use of this method, as well as the more general oriented bounding box (OBB) where the sides are orthogonal, while no longer having to be parallel to the coordinate axes (e.g., [30,56]). In addition, some data structures use other shapes for the bounding boxes such as spheres (e.g., SS-tree [51,98]), combinations of hyperrectangles and hyperspheres (e.g.,SR-tree [41]), truncated tetrahedra (e.g., prism tree [54]), as well as triangular pyramids which are 5-sided objects with two parallel triangular faces and three rectangular faces forming a three-dimensional pie slice (e.g., BOXTREE [11]). These data structures differ primarily in the properties of the bounding boxes, and their interrelationships, that they use to determine how to aggregate the bounding boxes, and, of course, the objects. Aggregation is an issue when the data structure is used in a dynamic environment, where objects are inserted and removed from the hierarchy thereby leading to elements that are full or sparse vis-a-vis the values of m and M.

As an example of an R-tree, consider the collection of straight line segment objects given in Figure 2(a) shown embedded in a 4×4 grid. Figure 2(b) is an example of the object hierarchy induced by an R-tree for this collection, with $m = 2$ and $M = 3$. Figure 2(c) shows the spatial extent of the bounding rectangles of the nodes in Figure 2(a), with heavy lines denoting the bounding rectangles corresponding to the leaf nodes, and broken lines denoting the bounding rectangles corresponding to the subtrees rooted at the nonleaf nodes.

The drawback of the object hierarchy approach is that from the perspective of a space decomposition method, the resulting hierarchy of bounding boxes often leads to a non-disjoint decomposition of the underlying space. This means that if a search fails to find an object in one path starting at the root, then it is not necessarily the case that the object will not be found in another path starting at the root. This is the case in Figure 2(c) when we search for the line segment object that contains Q. In particular, we first visit nodes R1 and R4 unsuccessfully, and thus need to visit nodes R2 and R5 in order to find the correct line segment object i.

The second method is based on a decomposition (usually recursive) of the underlying space into disjoint blocks so that a subset of the objects is associated with each block. There are several ways to proceed. The first is to simply redefine the decomposition and aggregation associated with the object hierarchy method so that the minimum bounding boxes are decomposed into disjoint boxes, thereby also implicitly partitioning the underlying objects that they bound. In this case, the partition of the underlying space is heavily dependent on the data and is said to be at arbitrary positions. The k-d-B-tree [58] and the R$^+$-tree [84] are examples of such an approach, with the difference being that in the k-d-B-tree, the entire space which

contains the objects is decomposed into subspaces and it is these subspaces that are aggregated, while in the R$^+$-tree, it is the bounding boxes that are decomposed and subsequently aggregated.

Figure 3 is an example of one possible R$^+$-tree for the collection of line segments in Figure 2(a). This particular tree is of order (2,3) although in general it is not possible to guarantee that all nodes save for the root node will always have a minimum of 2 entries. In particular, the expected B-tree performance guarantees are not necessarily valid (i.e., pages are not guaranteed to be m/M full) unless we are willing to perform very complicated record insertion and deletion procedures. Notice that in this example line segment objects c, h, and i appear in two different nodes. Of course, other variants are possible since the R$^+$-tree is not unique.

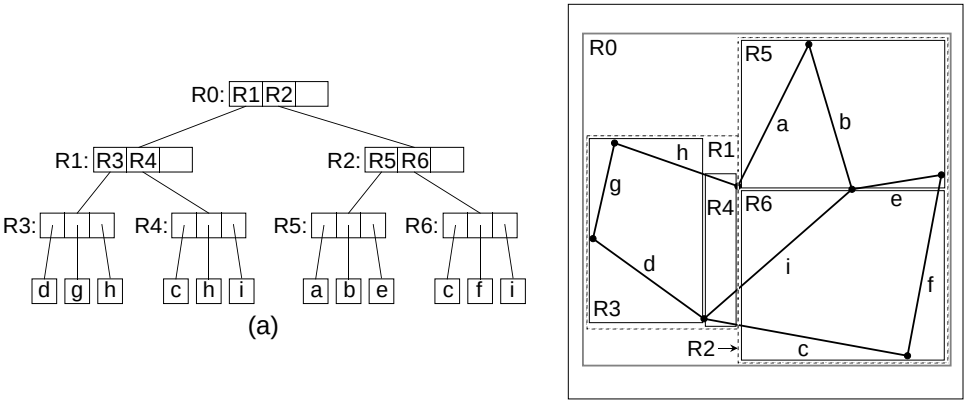


Figure 3: (a) R$^+$-tree for the collection of line segments in Figure 2 (a) with m=2 and M=3, and (b) the spatial extents of the bounding rectangles. Notice that the leaf nodes in the index also store bounding rectangles although this is only shown for the nonleaf nodes.

The second way is to partition the underlying space into cells (i.e., blocks) at fixed positions so that all resulting cells are of uniform size, which is the case when using the uniform grid (e.g., [13,45,59]), also the standard indexing method for maps. Figure 2(a) is an example of a 4×4 uniform grid in which a collection of straight line segments has been embedded. One drawback of the uniform grid is the possibility of a large number of empty or sparsely-filled cells when the objects are not uniformly distributed, as well as the possibility that most of the objects will lie in a small subset of the cells. This is resolved by making use of a variable resolution representation such as one of the quadtree variants (e.g., [76]) where the subset of the objects that are associated with the cells is defined by placing an upper bound on the number of objects that can be associated with each cell. The cells that comprise

the underlying space are recursively decomposed into congruent sibling cells whenever this upper bound is exceeded. Therefore, the upper bound serves as a *stopping condition* for the recursive decomposition process. An alternative, as exemplified by the PK-tree [75,95], makes use of a lower bound on the number of objects that can be associated with each cell (termed an *instantiation* or *aggregation* threshold). Depending on the underlying representation that is used, the result can also be viewed as a hierarchy of congruent cells (see, e.g., the pyramid structure [5,92], a family of representations that make use of multiple resolution and can be characterized as image hierarchies [76]).

The PR quadtree [52,74] is one example of a variable resolution representation for point objects where the underlying space in which a set of point objects lie is recursively decomposed into four equal-sized square-shaped cells until each cell is empty or contains just one object (i.e., the objects are sorted into the cells which act like bins). For example, Figure 4 is the PR quadtree for the set of point objects A–F and P. The PR quadtree represents the underlying decomposition as a tree although our figure only illustrates the resulting decomposition of the underlying space into cells (i.e., the leaf nodes/blocks of the PR quadtree).

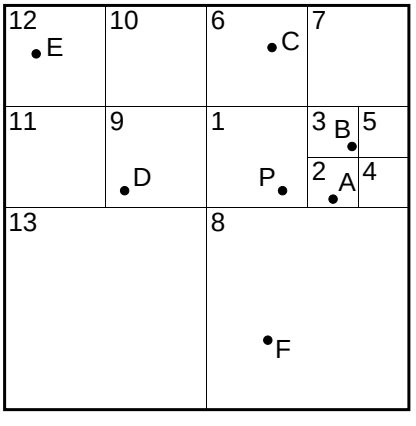
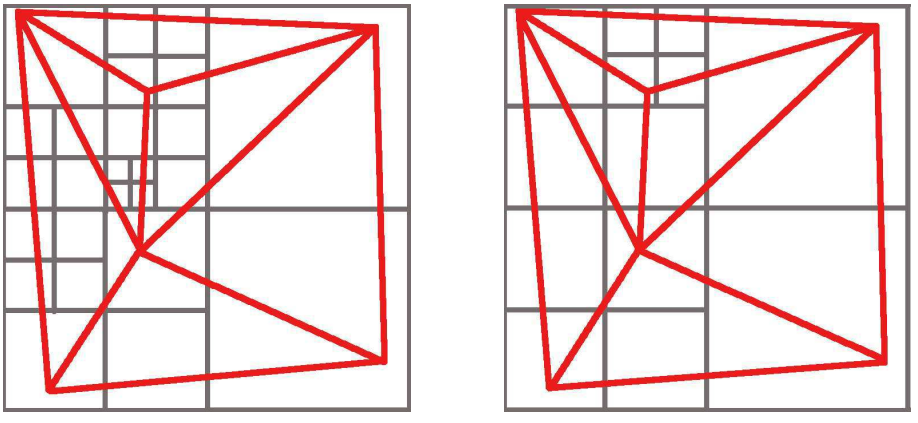


Figure 4: Block decomposition induced by the PR quadtree for the point objects A–F and P.

Turning to more complex such objects such as line segments, which have extent, we consider the PM_1 quadtree [81]. It is an example of a variable resolution representation for a collection of straight line segment objects such as the polygonal subdivision given in Figure 2(a). In this case, the stopping condition of its decomposition rule stipulates that partitioning occurs as long as a cell contains more than one line segment unless the line segments are all incident at the same vertex, which is also in the same cell (e.g., Figure 5(a)), The PM_1 quadtree and its variants are ideal for representing polygonal meshes as they provide an access

structure to enable the quick determination of the polygon that contains a given point (i.e., a point location operation). In particular, the PM_2 quadtree [81], which differs from the PM_1 quadtree by permitting a cell c to contain several line segments as long as they are incident at the same vertex v regardless of whether or not v is in c (e.g., Figure 5(b)), is particularly suitable for representing triangular meshes [20]. A similar representation to the PM_1 quadtree has been devised for collections of three-dimensional objects such as polyhedra images (e.g., [10] and the references cited in [76]). The decomposition criteria are such that no cell contains more than one face, edge, or vertex unless the faces all meet at the same vertex or are adjacent to the same edge. The above representations are said to be vertex-based. The bucket PM quadtree [76] (also termed a bucket PMR quadtree in [46]) and the PMR quadtree [49,50] are examples of edge-based representations where the decomposition criteria only involve the number of edges b (faces in three dimensions although the discussion below is in terms of two dimensions). In particular, in the former, the decomposition halts whenever a cell contains b or less edges while in the latter a cell is decomposed once and only once when it contains more than b edges. In this way, there is no need to split forever when b or more edges meet at a vertex.



(a) (b)

Figure 5: (a) PM_1 quadtree and (b) PM_2 quadtree for a collection of straight line segment objects that form a triangulation.

The above variants of the PM quadtree and PM octree represent an object by its boundary. The region quadtree [38,44] and region octree [37,48] are variable resolution representations of objects by their interiors. In particular, the environment containing the objects is recursively decomposed into four or eight, respectively, rectangular congruent blocks until each block is either completely occupied by an object or is empty. For example, Figure 6(b) is the block decomposition for the

region quadtree corresponding to the result of embedding the two-dimensional object in Figure 6(a) in an 8×8 grid, while Figure 7(b) is the block decomposition for the region octree corresponding to the three-dimensional staircaselike object in Figure 7(a).

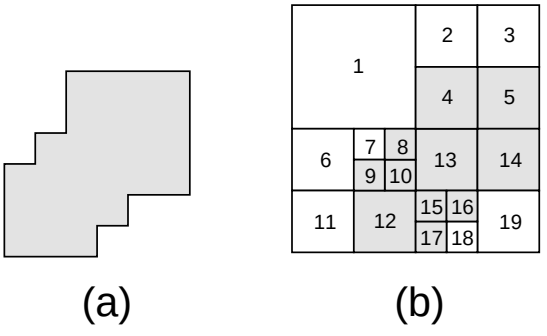


Figure 6: (a) Sample object, and (b) its region quadtree block decomposition with the blocks of the object being shaded, assuming that it is embedded in an 8×8 grid.

Region Quadtrees and octrees make it easy to implement algorithms for a number of basic operations in computer graphics, image processing, as well as numerous other applications (e.g., [70]). etc. In particular, algorithms have been devised for converting between region quadtrees and numerous representations such as binary arrays [60], boundary codes [24,61], rasters [62,68,86], medial axis transforms [67,69], and terrain models [88], as well as for many standard operations such as connected component labeling [64], perimeters [63], distance [65], image dilation [4], and computing Euler numbers [23]. Algorithms have also been devised for converting between region octrees and boundary models [91] and constructive solid geometry (CSG) [80].

Region octrees are also known as volumetric or voxel representations and are useful for medical applications. They are to be contrasted with procedural representations such as constructive solid geometry (CSG) [57] where primitive instances of objects are combined to form more complex objects by use of geometric transformations and regularized Boolean set operations (e.g., union, intersection). A disadvantage of the CSG representation is that it is not unique. In particular, there are frequently several ways of constructing an object (e.g., from different primitive elements). In addition, there is no overall notion of geometry except of the primitives that form each of the objects and thus there is no easy correlation between the objects and the space in which they are embedded unless techniques such as the PM-CSG tree [99] are used.

The principal drawback of the disjoint method is that when the objects have extent (e.g., line segments, rectangles, and any other non-point objects), then an object is associated with more than one cell when the object has been decomposed.

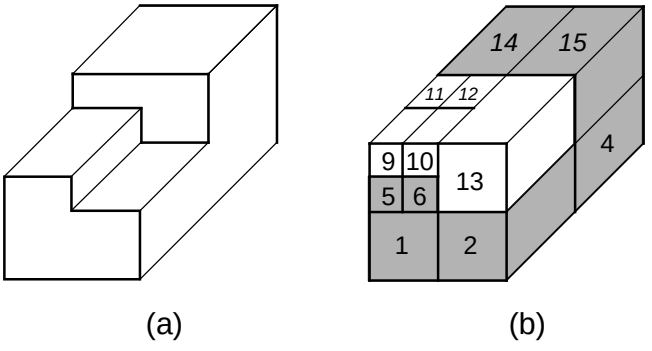


Figure 7: (a) Example three-dimensional object, and (b) its region octree block decomposition.

This means that queries such as those that seek the length of all objects in a particular spatial region will have to remove duplicate objects before reporting the total length. Nevertheless, methods have been developed that avoid these duplicates by making use of the geometry of the type of the data that is being represented (e.g., [6,7,21]). Note that the result of constraining the positions of the partitions means that there is a limit on the possible sizes of the resulting cells (e.g., a power of 2 in the case of a quadtree variant). However, the result is that the underlying representation is good for operations between two different data sets (e.g., a spatial join [36,40]) as their representations are in registration (i.e., it is easy to correlate occupied and unoccupied space in the two data sets, which is not easy when the positions of the partitions are not constrained as is the case with methods rooted in representations based an object hierarchy even though the resulting decomposition of the underlying space is disjoint). For a recent empirical comparison of these representations with respect to multidimensional point data, see [43].

The PR, PM, and region quadtrees make use of a space hierarchy of where each level of the hierarchy contains congruent cells. The difference is that in the PR quadtree, each object is associated with just one cell, while in the PM and region quadtrees, the extent of the objects causes them to be decomposed into subobjects and thereby possibly be associated with more than one cell, although the cells are disjoint. At times, we want to use a space decomposition method that makes use of a hierarchy of congruent cells while still not decomposing the objects. In this case, we relax the disjointness requirement by stipulating that only the cells at a given level (i.e., depth) of the hierarchy must be disjoint. In particular, we recursively decompose the cells that comprise the underlying space into congruent sibling cells so that each object is associated with just one cell, and this is the smallest possible congruent cell that contains the object in its entirety. Assuming a top-down subdivision process that decomposes each cell into four square cells (i.e., a quadtree) at each level of decomposition, the result is that each object is

associated with its minimum enclosing quadtree cell. Subdivision ceases whenever a cell contains no objects. Alternatively, subdivision can also cease once a cell is smaller than a predetermined threshold size. This threshold is often chosen to be equal to the expected size of the objects. We use the term *MX-CIF quadtree* [1,42] (see also the multilayer grid file [89], R-file [39], filter tree [85], and SQ-histogram [3]) to describe such a decomposition method.

In order to simplify our presentation, we assume that the objects stored in the MX-CIF quadtree are rectangles, although the MX-CIF quadtree is applicable to arbitrary objects in arbitrary dimensions in which case it keeps track of their minimum bounding boxes. For example, Figure 8(b) is the tree representation of the MX-CIF quadtree for a collection of rectangle objects given in Figure 8(a). Note that objects can be associated with both terminal and non-terminal nodes of the tree.

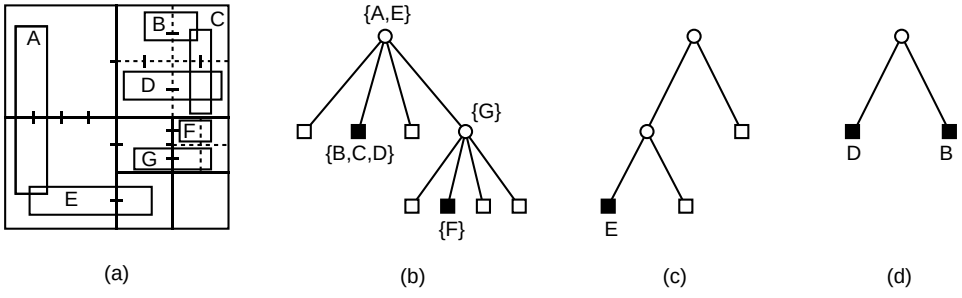


Figure 8: (a) Collection of rectangle objects and the cell decomposition induced by the MX-CIF quadtree; (b) the tree representation of (a); the binary trees for the y axes passing through the root of the tree in (b), and through (d) the NE son of the root of the tree in (b).

Since there is no limit on the number of objects that are associated with a particular cell, an additional decomposition rule is sometimes provided to distinguish between these objects. For example, in the case of the MX-CIF quadtree, a one-dimensional analog of the two-dimensional decomposition rule is used. In particular, all objects that are associated with a given cell b are partitioned into two sets: those that intersect (or whose sides are collinear) with the vertical axis passing through the center of b, and those that intersect (or whose sides are collinear) with the horizontal axis passing through the center of b. Objects that intersect with the center of b are associated with the horizontal axis. Associated with each axis is a one-dimensional MX-CIF quadtree (i.e., a binary tree), where each object o is associated with the node that corresponds to o's minimum enclosing interval. For example, Figure 8(c) and Figure 8(d) illustrate the binary trees associated with the y axes passing through the root and the NE son of the root, respectively, of the MX-CIF quadtree of Figure 8(b). Thus we see that the two-dimensional MX-CIF

quadtree acts like a hashing function with the one-dimensional MX-CIF quadtree playing the role of a collision resolution technique.

The MX-CIF quadtree can be interpreted as an object hierarchy where the objects appear at different levels of the hierarchy and the congruent cells play the same role as the minimum bounding boxes. The difference is that the set of possible minimum bounding boxes is constrained to the set of possible congruent cells. Thus, we can view the MX-CIF quadtree as a variable resolution R-tree. An alternative interpretation is that the MX-CIF quadtree provides a variable number of grids, each one being at half the resolution of its immediate successor, where an object is associated with the grid whose cells have the tightest fit. In fact, this interpretation forms the basis of the *filter tree* [85] and the *multilayer grid file* [89], where the only difference from the MX-CIF quadtree is the nature of the access structure for the cells (i.e., a hierarchy of grids based on a regular decomposition for the filter tree and based on a grid file for the multilayer grid file, and a tree structure for the MX-CIF quadtree).

One of the main drawbacks of the MX-CIF quadtree is that the size (i.e., width w) of the cell c corresponding to the minimum enclosing quadtree cell of object o's minimum enclosing bounding box b is not a function of the size of b or o. Instead, it is dependent on the position of o. In fact, c is often considerably larger than b thereby causing inefficiency in search operations due to a reduction in the ability to prune objects from further consideration. This situation arises whenever b overlaps the axes lines that pass through the center of c, and thus w can be as large as the width of the entire underlying space.

There are several ways of overcoming this drawback. One easy way is to introduce redundancy (i.e., representing the object several times thereby replicating the number of references to it) by decomposing the quadtree cell c into smaller quadtree cells, each of which minimally encloses some portion of o (or, alternatively, some portion of o's minimum enclosing bounding box b) and contains a reference to o. The expanded MX-CIF quadtree [2] is a simple example of such an approach where c is decomposed once into four subblocks c_i, which are then decomposed further until obtaining the minimum enclosing quadtree cell s_i for the portion of o, if any, that is covered by c_i. A more general approach. used in spatial join algorithms [40], sets a bound on the number of replications, (termed a *size bound* [53] and used in the GESS method [22]) or on the size of the covering quadtree cells resulting from the decomposition of c that contain the replicated references (termed an *error bound* [53]).

Replicating the number of references to the objects is reminiscent of the manner in which the non-disjointness of the decomposition of the underlying space resulting from the use of an object hierarchy was overcome, and thus has the same shortcoming of possibly requiring the application of a duplicate object removal step prior to reporting the answer to some queries. The *cover fieldtree* [26,27], and the equivalent *loose quadtree* (*loose octree* in three dimensions) [94], adopt a different approach at overcoming the independence of the sizes of c and b drawback. In particular, they do not replicate the objects. Instead, they expand the size of the space that is

spanned by each quadtree cell c of width w by a cell expansion factor p ($p > 0$) so that the expanded cell is of width $(1 + p) \cdot w$. In this case, an object is associated with its minimum enclosing expanded quadtree cell. It has been shown that given a quadtree cell c of width w and cell expansion factor p, the radius r of the minimum bounding box b of the smallest object o that could possibly be associated with c must be greater than $pw/4$ [94]. However, the utility of the loose quadtree is best evaluated in terms of the inverse of this relation (i.e., the maximum possible width w of c given an object o with minimum bounding box b of radius r) as reducing w is the primary motivation for the development of the loose quadtree as an alternative to the MX-CIF quadtree.

It has been shown [78] that the maximum possible width w of c given an object o with minimum bounding box b of radius r is just a function of r and p and is independent of the position of o. More precisely, taking the ratio of cell to bounding box width $w/(2r)$, we have [78]:

$$1/(1 + p) \leq w/(2r) \leq 1/p.$$

In particular, the range of possible ratios of width $w/(2r)$ as a function of p for $p \geq 1$ takes on at most two values, and usually just one value [78].

The ideal value for p is 1 [94]. The rationale is that using cell expansion factors much smaller than 1 increases the likelihood that the minimum enclosing expanded quadtree cell is large (as is the case for the MX-CIF quadtree, where $p = 0$), and that letting p be much larger than 1 results in the areas spanned by the expanded quadtree cells being too large, thereby having much overlap. For example, letting $p = 1$, Figure 9 is the loose quadtree corresponding to the collection of objects in Figure 8(a) and its MX-CIF quadtree in Figure 8(b). In this example, there are only two differences between the loose and MX-CIF quadtrees:

(1) Rectangle object E is associated with the SW child of the root of the loose quadtree instead of with the root of the MX-CIF quadtree.
(2) Rectangle object B is associated with the NW child of the NE child of the root of the loose quadtree instead of with the NE child of the root of the MX-CIF quadtree.

Note that the loose quadtree (cover fieldtree) is not the only approach at overcoming the drawback of the MX-CIF quadtree. In particular, the partition fieldtree [26,27] is an alternative method of overcoming the drawback of the MX-CIF quadtree. The partition fieldtree proceeds by shifting the positions of the centroids of cells at successive levels of subdivision by one-half the width of the cell that is being subdivided. Figure 10 shows an example of such a subdivision. This subdivision rule guarantees that the width w of the minimum enclosing quadtree cell for the minimum bounding box b for object o is bounded by eight times the maximum extent r of b [27,76]. The same ratio is obtained for the cover fieldtree when $p = 1/4$, and thus the partition fieldtree is superior to the cover fieldtree when $p < 1/4$ [76].

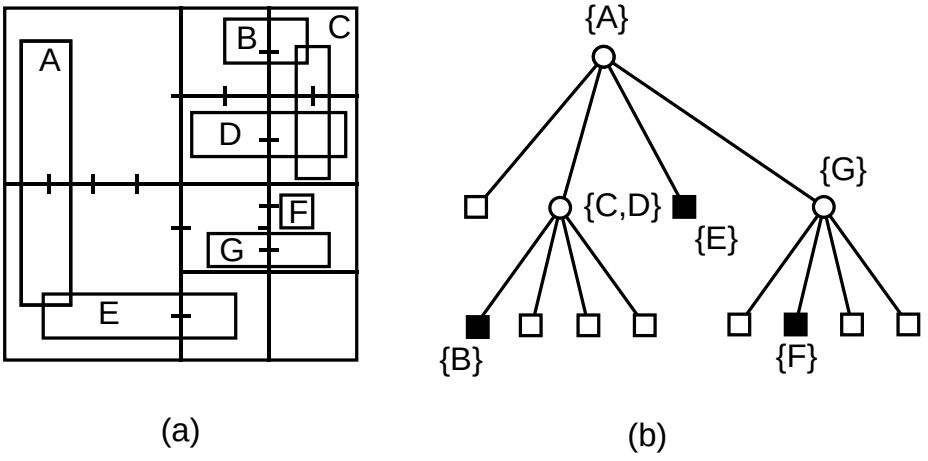


Figure 9: (a) Cell decomposition induced by the loose quadtree for a collection of rectangle objects identical to those in Figure 8(a), and (b) its tree representation.

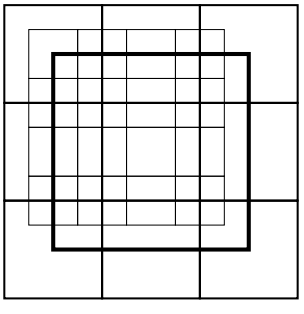


Figure 10: Example of the subdivision induced by a partition fieldtree.

3. Examples of the Utility of Sorting

As an example of the utility of sorting spatial data suppose that we want to determine the nearest object to a given point (i.e., a "pick" operation in computer graphics). In order to see how the search is facilitated by sorting the underlying data, consider the set of point objects A–F in Figure 4 which are stored in a PR quadtree [52,74], and let us find the nearest neighbor of P. The search must first determine the leaf that contains the location/object whose nearest neighboring object is sought (i.e., P). Assuming a tree-based index, this is achieved by a top-down recursive algorithm. Initially, at each level of the recursion, we explore the subtree that contains P. Once the leaf node containing P has been found (i.e., 1), the distance from P to the nearest object in the leaf node is calculated (empty leaf nodes have a value of infinity). Next, we unwind the recursion so that at each level, we

search the subtrees that represent regions overlapping a circle centered at P whose radius is the distance to the closest object that has been found so far. When more than one subtree must be searched, the subtrees representing regions nearer to P are searched before the subtrees that are farther away (since it is possible that an object in them might make it unnecessary to search the subtrees that are farther away).

In our example, the order in which the nodes are visited is given by their labels. We visit the brothers of the node 1 containing the query point P (and all remaining nodes at each level) in the order of the minimum distance from P to their borders (i.e., SE, NW, and NE for node 1). Therefore, as we unwind for the first time, we visit the eastern brother of node 1 and its subtrees (nodes 2 and 3 followed by nodes 4 and 5), node 6, and node 7. Note that once we have visited node 2, there is no need to visit node 4 since node 2 contains A. However, we must still visit node 3 containing point B (closer than A), but now there is no need to visit node 5. Similarly, there is no need to visit nodes 6 and 7 as they are too far away from P given our knowledge of A. Unwinding one more level reveals that due to the distance between P and A, we must visit node 8 as it could contain a point that is closer to P than A; however, there is no need to visit nodes 9, 10, 11, 12, and 13.

The algorithm that we described can also be adapted to find the k nearest neighbors in which case the pruning of objects that cannot serve as the k nearest neighbors is achieved by making use of the distance to the kth nearest object that has been found so far. Having retrieved the k closest objects, should we be interested in retrieving an additional object (i.e., the $k+1$th nearest object), then we have to reinvoke the algorithm to find the $k+1$ nearest objects. An alternative approach is incremental and makes use of a priority queue [32,33,34] so that there is no need to look again for the neighboring objects that have been reported so far. Note that although in the above proximity is measured in terms of as "the crow flies", these methods can also be used to support finding nearest neighbors in a graph such as a road network (e.g., [79,82]).

There are many other applications where the sorting of objects is useful, and below we review a few that arise in computer graphics. For example, sorting forms the basis of all operations on z buffers, visibility calculations (e.g., BSP trees [28]), as well as back-to-front and front-to-back display algorithms. Sorting also forms the basis of Warnock's hidden-line [96] and hidden-surface [97] algorithms that repeatedly subdivide the picture area into successively smaller blocks while simultaneously searching it for areas that are sufficiently simple to be displayed. In addition, sorting is used to accelerate ray tracing by finding ray-object intersections (e.g., [9]) using neighbor finding techniques [66,72] for quadtrees and octrees [29,71].

4. Concluding Remarks

An overview has been given of the rationale for sorting spatial objects in order to be able to index them thereby facilitating a number of operations involving

search in the multidimensional domain. A distinction has been made between spatial objects that could be represented by traditional methods that have been applied to point data and those that have extent thereby rendering the traditional methods inapplicable.

Sorting is also used as the basis of an index in an environment where the data is drawn from a metric space rather than a vector space. In this case, the only information that we have is a distance function d (often a matrix) that indicates the degree of similarity (or dissimilarity) between all pairs of objects, given a set of N objects. Usually, it is required that d obey the triangle inequality, be nonnegative, and be symmetric, in which case it is known as a *metric* and also referred to as a *distance metric*. Indexes in such an environment are based on either picking one distinguished object p and a value r, and then recursively subdividing the remaining objects into two classes depending on a comparison of their distance from p with r, or by choosing two distinguished objects p_1 and p_2 and recursively subdividing the remaining objects into two classes depending on which of p_1 or p_2 is closer (e.g., [76,93]). The difference between these methods and those for data that lies in a vector space is that the subdivision lines in the embedding space from which the objects are drawn are explicit for the vector space while they are implicit for the metric space (see [76] for more details).

The functioning of these various spatial sorting methods can be experienced by trying VASCO [14,15,16,18], a system for Visualizing and Animating Spatial Constructs and Operations. VASCO consists of a set of spatial index JAVA™ (e.g., [8]) applets that enable users on the worldwide web to experiment with a number of hierarchical representations (e.g., [73,74,76]) for different spatial data types, and see animations of how they support a number of search queries (e.g., nearest neighbor and range queries). The VASCO system can be found at `http://cs.umd.edu/~hjs/quadtree/`. For an example of their use in a spatial database/geographic information system (GIS), see the SAND Spatial Browser [17,25,77] and the QUILT system [87]. Such systems find use in a number of alternative application domains (e.g., digital government [47]).

Acknowledgments

I am deeply grateful to Jagan Sankaranarayanan for help with the figures and preparation for publication.

References

1. D. J. Abel and J. L. Smith. A data structure and algorithm based on a linear key for a rectangle retrieval problem. *Computer Vision, Graphics, and Image Processing*, 24(1):1–13, Oct. 1983.
2. D. J. Abel and J. L. Smith. A data structure and query algorithm for a database of areal entities. *Australian Computer Journal*, 16(4):147–154, Nov. 1984.
3. A. Aboulnaga and J. F. Naughton. Accurate estimation of the cost of spatial selec-

tions. In *Proceedings of the 16th IEEE International Conference on Data Engineering*, pages 123–134, San Diego, CA, Feb. 2000.

4. C.-H. Ang, H. Samet, and C. A. Shaffer. A new region expansion for quadtrees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(7):682–686, July 1990. Also see *Proceedings of the Third International Symposium on Spatial Data Handling*, pages 19–37, Sydney, Australia, August 1988.
5. W. G. Aref and H. Samet. Efficient processing of window queries in the pyramid data structure. In *Proceedings of the 9th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 265–272, Nashville, TN, Apr. 1990. Also in *Proceedings of the Fifth Brazilian Symposium on Databases*, pages 15–26, Rio de Janeiro, Brazil, April 1990.
6. W. G. Aref and H. Samet. Uniquely reporting spatial objects: yet another operation for comparing spatial data structures. In *Proceedings of the 5th International Symposium on Spatial Data Handling*, pages 178–189, Charleston, SC, Aug. 1992.
7. W. G. Aref and H. Samet. Hashing by proximity to process duplicates in spatial databases. In *Proceedings of the 3rd International Conference on Information and Knowledge Management (CIKM)*, pages 347–354, Gaithersburg, MD, Dec. 1994.
8. K. Arnold and J. Gosling. *The JAVA*$^{\text{TM}}$ *Programming Language*. Addison-Wesley, Reading, MA, 1996.
9. J. Arvo and D. Kirk. A survey of ray tracing acceleration techniques. In A. S. Glassner, editor, *An Introduction to Ray Tracing*, chapter 6, pages 201–262. Academic Press, New York, 1989.
10. D. Ayala, P. Brunet, R. Juan, and I. Navazo. Object representation by means of nonminimal division quadtrees and octrees. *ACM Transactions on Graphics*, 4(1):41–59, Jan. 1985.
11. G. Barequet, B. Chazelle, L. J. Guibas, J. S. B. Mitchell, and A. Tal. BOXTREE: a hierarchical representation for surfaces in 3D. In J. Rossignac and F. X. Sillion, editors, *Proceedings of the EUROGRAPHICS'96 Conference*, pages 387–396, 484, Poitiers, France, Aug. 1996. Also in *Computer Graphics Forum*, 15(3):387–396, 484, August 1996.
12. N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: an efficient and robust access method for points and rectangles. In *Proceedings of the ACM SIGMOD Conference*, pages 322–331, Atlantic City, NJ, June 1990.
13. J. L. Bentley and J. H. Friedman. Data structures for range searching. *ACM Computing Surveys*, 11(4):397–409, Dec. 1979.
14. F. Brabec and H. Samet. The VASCO R-tree JAVA$^{\text{TM}}$ applet. In Y. Ioannidis and W. Klas, editors, *Visual Database Systems (VDB4). Proceedings of the IFIP TC2//WG2.6 Fourth Working Conference on Visual Database Systems*, pages 147–153, L'Aquila, Italy, May 1998. Chapman and Hall.
15. F. Brabec and H. Samet. Visualizing and animating R-trees and spatial operations in spatial databases on the worldwide web. In Y. Ioannidis and W. Klas, editors, *Visual Database Systems (VDB4). Proceedings of the IFIP TC2//WG2.6 Fourth Working Conference on Visual Database Systems*, pages 123–140, L'Aquila, Italy, May 1998. Chapman and Hall.
16. F. Brabec and H. Samet. Visualizing and animating search operations on quadtrees on the worldwide web. In K. Kedem and M. Katz, editors, *Proceedings of the 16th European Workshop on Computational Geometry*, pages 70–76, Eilat, Israel, Mar. 2000.
17. F. Brabec and H. Samet. Client-based spatial browsing on the world wide web. *IEEE Internet Computing*, 11(1):52–59, January/February 2007.

18. F. Brabec, H. Samet, and C. Yilmaz. VASCO: visualizing and animating spatial constructs and operations. In *Proceedings of the 19th Annual Symposium on Computational Geometry*, pages 374–375, San Diego, CA, June 2003.
19. D. Comer. The ubiquitous B-tree. *ACM Computing Surveys*, 11(2):121–137, June 1979.
20. L. De Floriani, M. Facinoli, P. Magillo, and D. Dimitri. A hierarchical spatial index for triangulated surfaces. In J. Braz, N. Jardim Nunes, and J. Madeiras Pereira, editors, *Proceedings of the Third International Conference on Computer Graphics Theory and Applications (GRAPP 2008)*, pages 86–91, Funchal, Madeira, Portugal, Jan. 2008.
21. J.-P. Dittrich and B. Seeger. Data redundancy and duplicate detection in spatial join processing. In *Proceedings of the 16th IEEE International Conference on Data Engineering*, pages 535–546, San Diego, CA, Feb. 2000.
22. J.-P. Dittrich and B. Seeger. GESS: a scalable similarity-join algorithm for mining large data sets in high dimensional spaces. In *Proceedings of the 7th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 47–56, San Francisco, Aug. 2001.
23. C. R. Dyer. Computing the Euler number of an image from its quadtree. *Computer Graphics and Image Processing*, 13(3):270–276, July 1980. Also University of Maryland Computer Science Technical Report TR–769, May 1979.
24. C. R. Dyer, A. Rosenfeld, and H. Samet. Region representation: boundary codes from quadtrees. *Communications of the ACM*, 23(3):171–179, Mar. 1980. Also University of Maryland Computer Science Technical Report TR–732, February 1979.
25. C. Esperança and H. Samet. Experience with SAND/Tcl: a scripting tool for spatial databases. *Journal of Visual Languages and Computing*, 13(2):229–255, Apr. 2002.
26. A. Frank. Problems of realizing LIS: storage methods for space related data: the fieldtree. Technical Report 71, Institute for Geodesy and Photogrammetry, ETH, Zurich, Switzerland, June 1983.
27. A. U. Frank and R. Barrera. The Fieldtree: a data structure for geographic information systems. In A. Buchmann, O. Günther, T. R. Smith, and Y.-F. Wang, editors, *Design and Implementation of Large Spatial Databases—1st Symposium, SSD'89*, vol. 409 of Springer-Verlag Lecture Notes in Computer Science, pages 29–44, Santa Barbara, CA, July 1989.
28. H. Fuchs, Z. M. Kedem, and B. F. Naylor. On visible surface generation by a priori tree structures. *Computer Graphics*, 14(3):124–133, July 1980. Also in *Proceedings of the SIGGRAPH'80 Conference*, Seattle, WA, July 1980.
29. A. S. Glassner. Space subdivision for fast ray tracing. *IEEE Computer Graphics and Applications*, 4(10):15–22, Oct. 1984.
30. S. Gottschalk, M. C. Lin, and D. Manocha. OBBTree: a hierarchical structure for rapid interference detection. In *Proceedings of the SIGGRAPH'96 Conference*, pages 171–180, New Orleans, LA, Aug. 1996.
31. A. Guttman. R-trees: a dynamic index structure for spatial searching. In *Proceedings of the ACM SIGMOD Conference*, pages 47–57, Boston, June 1984.
32. A. Henrich. A distance-scan algorithm for spatial access structures. In N. Pissinou and K. Makki, editors, *Proceedings of the 2nd ACM Workshop on Geographic Information Systems*, pages 136–143, Gaithersburg, MD, Dec. 1994.
33. G. R. Hjaltason and H. Samet. Ranking in spatial databases. In M. J. Egenhofer and J. R. Herring, editors, *Advances in Spatial Databases—4th International Symposium, SSD'95*, vol. 951 of Springer-Verlag Lecture Notes in Computer Science, pages 83–95, Portland, ME, Aug. 1995.
34. G. R. Hjaltason and H. Samet. Distance browsing in spatial databases. *ACM Trans-*

actions on Database Systems, 24(2):265–318, June 1999. Also University of Maryland Computer Science Technical Report TR–3919, July 1998.
35. E. G. Hoel and H. Samet. Efficient processing of spatial queries in line segment databases. In O. Günther and H.-J. Schek, editors, *Advances in Spatial Databases—2nd Symposium, SSD'91*, vol. 525 of Springer-Verlag Lecture Notes in Computer Science, pages 237–256, Zurich, Switzerland, Aug. 1991.
36. E. G. Hoel and H. Samet. Benchmarking spatial join operations with spatial output. In U. Dayal, P. M. D. Gray, and S. Nishio, editors, *Proceedings of the 21st International Conference on Very Large Data Bases (VLDB)*, pages 606–618, Zurich, Switzerland, Sept. 1995.
37. G. M. Hunter. *Efficient computation and data structures for graphics*. PhD thesis, Department of Electrical Engineering and Computer Science, Princeton University, Princeton, NJ, 1978.
38. G. M. Hunter and K. Steiglitz. Operations on images using quad trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1(2):145–153, Apr. 1979.
39. A. Hutflesz, H.-W. Six, and P. Widmayer. The R-file: an efficient access structure for proximity queries. In *Proceedings of the 6th IEEE International Conference on Data Engineering*, pages 372–379, Los Angeles, Feb. 1990.
40. E. Jacox and H. Samet. Spatial join techniques. *ACM Transactions on Database Systems*, 32(1):7, Mar. 2007. Also an expanded version in University of Maryland Computer Science Technical Report TR–4730, June 2005.
41. N. Katayama and S. Satoh. The SR-tree: an index structure for high-dimensional nearest neighbor queries. In J. Peckham, editor, *Proceedings of the ACM SIGMOD Conference*, pages 369–380, Tucson, AZ, May 1997.
42. G. Kedem. The quad-CIF tree: a data structure for hierarchical on-line algorithms. In *Proceedings of the 19th Design Automation Conference*, pages 352–357, Las Vegas, NV, June 1982. Also University of Rochester Computer Science Technical Report TR–91, September 1981.
43. Y. J. Kim and J. M. Patel. Rethinking choices for multi-dimensional point indexing: making the case for the often ignored quadtree. In *Proceedings of the Third Biennial Conference on Innovative Data Systems Research (CIDR 2007)*, pages 281–291, Asilomar, CA, Jan. 2007.
44. A. Klinger. Patterns and search statistics. In J. S. Rustagi, editor, *Optimizing Methods in Statistics*, pages 303–337. Academic Press, New York, 1971.
45. D. E. Knuth. *The Art of Computer Programming: Sorting and Searching*, volume 3. Addison-Wesley, Reading, MA, second edition, 1998.
46. M. Lindenbaum, H. Samet, and G. R. Hjaltason. A probabilistic analysis of trie-based sorting of large collections of line segments in spatial databases. *SIAM Journal on Computing*, 35(1):22–58, Sept. 2005. Also see *Proceedings of the 10th International Conference on Pattern Recognition*, vol. II, pages 91–96, Atlantic City, NJ, June 1990 and University of Maryland Computer Science Technical Report TR–3455.1, February 2000.
47. G. Marchionini, H. Samet, and L. Brandt. Introduction to the digital government special issue. *Communications of the ACM*, 46(1):24–27, Jan. 2003.
48. D. Meagher. Geometric modeling using octree encoding. *Computer Graphics and Image Processing*, 19(2):129–147, June 1982.
49. R. C. Nelson and H. Samet. A consistent hierarchical representation for vector data. *Computer Graphics*, 20(4):197–206, Aug. 1986. Also in *Proceedings of the SIGGRAPH'86 Conference*, Dallas, TX, August 1986.
50. R. C. Nelson and H. Samet. A population analysis for hierarchical data structures.

In *Proceedings of the ACM SIGMOD Conference*, pages 270–277, San Francisco, May 1987.

51. S. M. Omohundro. Five balltree construction algorithms. Technical Report TR–89–063, International Computer Science Institute, Berkeley, CA, Dec. 1989.
52. J. A. Orenstein. Multidimensional tries used for associative searching. *Information Processing Letters*, 14(4):150–157, June 1982.
53. J. A. Orenstein. Redundancy in spatial databases. In *Proceedings of the ACM SIGMOD Conference*, pages 294–305, Portland, OR, June 1989.
54. J. Ponce and O. Faugeras. An object centered hierarchical representation for 3d objects: the prism tree. *Computer Vision, Graphics, and Image Processing*, 38(1):1–28, Apr. 1987.
55. F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, New York, 1985.
56. D. R. Reddy and S. Rubin. Representation of three-dimensional objects. Computer Science Technical Report CMU–CS–78–113, Carnegie-Mellon University, Pittsburgh, PA, Apr. 1978.
57. A. A. G. Requicha and H. B. Voelcker. Solid modeling: a historical summary and contemporary assessment. *IEEE Computer Graphics and Applications*, 2(2):9–24, Mar. 1982.
58. J. T. Robinson. The K-D-B-tree: a search structure for large multidimensional dynamic indexes. In *Proceedings of the ACM SIGMOD Conference*, pages 10–18, Ann Arbor, MI, Apr. 1981.
59. J. B. Rothnie Jr. and T. Lozano. Attribute based file organization in a paged memory environment. *Communications of the ACM*, 17(2):63–69, Feb. 1974.
60. H. Samet. Region representation: quadtrees from binary arrays. *Computer Graphics and Image Processing*, 13(1):88–93, May 1980. Also University of Maryland Computer Science Technical Report TR–767, May 1979.
61. H. Samet. Region representation: quadtrees from boundary codes. *Communications of the ACM*, 23(3):163–170, Mar. 1980. Also University of Maryland Computer Science Technical Report TR–741, March 1979.
62. H. Samet. An algorithm for converting rasters to quadtrees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 3(1):93–95, Jan. 1981. Also University of Maryland Computer Science Technical Report TR–766, May 1979.
63. H. Samet. Computing perimeters of images represented by quadtrees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 3(6):683–687, Nov. 1981. Also University of Maryland Computer Science Technical Report TR–755, April 1979.
64. H. Samet. Connected component labeling using quadtrees. *Journal of the ACM*, 28(3):487–501, July 1981. Also University of Maryland Computer Science Technical Report TR–756, April 1979.
65. H. Samet. Distance transform for images represented by quadtrees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 4(3):298–303, May 1982. Also University of Maryland Computer Science Technical Report TR–780, July 1979.
66. H. Samet. Neighbor finding techniques for images represented by quadtrees. *Computer Graphics and Image Processing*, 18(1):37–57, Jan. 1982. Also in *Digital Image Processing and Analysis: Vol. 2: Digital Image Analysis*, R. Chellappa and A. Sawchuck, eds., pages 399–419, IEEE Computer Society Press, Washington, DC, 1986; and University of Maryland Computer Science Technical Report TR–857, January 1980.
67. H. Samet. A quadtree medial axis transform. *Communications of the ACM*, 26(9):680–693, Sept. 1983. Also see CORRIGENDUM, *Communications of the ACM*, 27(2):151,

February 1984 and University of Maryland Computer Science Technical Report TR–803, August 1979.
68. H. Samet. Algorithms for the conversion of quadtrees to rasters. *Computer Vision, Graphics, and Image Processing*, 26(1):1–16, Apr. 1984. Also University of Maryland Computer Science Technical Report TR–979, November 1980.
69. H. Samet. Reconstruction of quadtrees from quadtree medial axis transforms. *Computer Vision, Graphics, and Image Processing*, 29(3):311–328, Mar. 1985. Also University of Maryland Computer Science Technical Report TR–1224, October 1982.
70. H. Samet. An overview of quadtrees, octrees, and related hierarchical data structures. In R. A. Earnshaw, editor, *Theoretical Foundations of Computer Graphics and CAD*, vol. 40 of NATO ASI Series F: Computer and System Sciences, pages 51–68. Springer-Verlag, Berlin, West Germany, 1988.
71. H. Samet. Implementing ray tracing with octrees and neighbor finding. *Computers & Graphics*, 13(4):445–460, 1989. Also University of Maryland Computer Science Technical Report TR–2204, February 1989.
72. H. Samet. Neighbor finding in images represented by octrees. *Computer Vision, Graphics, and Image Processing*, 46(3):367–386, June 1989. Also University of Maryland Computer Science Technical Report TR–1968, January 1988.
73. H. Samet. *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS*. Addison-Wesley, Reading, MA, 1990.
74. H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, Reading, MA, 1990.
75. H. Samet. Decoupling partitioning and grouping: overcoming shortcomings of spatial indexing with bucketing. *ACM Transactions on Database Systems*, 29(4):789–830, Dec. 2004. Also University of Maryland Computer Science Technical Report TR–4523, August 2003.
76. H. Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan-Kaufmann, San Francisco, 2006.
77. H. Samet, H. Alborzi, F. Brabec, C. Esperança, G. R. Hjaltason, F. Morgan, and E. Tanin. Use of the SAND spatial browser for digital government applications. *Communications of the ACM*, 46(1):63–66, Jan. 2003.
78. H. Samet and J. Sankaranarayanan. Maximum containing cell sizes in cover field-trees and loose quadtrees and octrees. Computer Science Technical Report TR–4900, University of Maryland, College Park, MD, Oct. 2007.
79. H. Samet, J. Sankaranarayanan, and H. Alborzi. Scalable network distance browsing in spatial databases. In *Proceedings of the ACM SIGMOD Conference*, pages 43–54, Vancouver, Canada, June 2008. Also University of Maryland Computer Science Technical Report TR–4865, April 2007 (SIGMOD 2008 Best Paper Award).
80. H. Samet and M. Tamminen. Bintrees, CSG trees, and time. *Computer Graphics*, 19(3):121–130, July 1985. Also in *Proceedings of the SIGGRAPH'85 Conference*, San Francisco, July 1985.
81. H. Samet and R. E. Webber. Storing a collection of polygons using quadtrees. *ACM Transactions on Graphics*, 4(3):182–222, July 1985. Also see *Proceedings of Computer Vision and Pattern Recognition'83*, pages 127–132, Washington, DC, June 1983 and University of Maryland Computer Science Technical Report TR–1372, February 1984.
82. J. Sankaranarayanan, H. Alborzi, and H. Samet. Efficient query processing on spatial networks. In *Proceedings of the 13th ACM International Symposium on Advances in Geographic Information Systems*, pages 200–209, Bremen, Germany, Nov. 2005.
83. B. Seeger and H.-P. Kriegel. Techniques for design and implementation of efficient spatial access methods. In F. Bachillon and D. J. DeWitt, editors, *Proceedings of the*

14th International Conference on Very Large Databases (VLDB), pages 360–371, Los Angeles, Aug. 1988.

84. T. Sellis, N. Roussopoulos, and C. Faloutsos. The R^+-tree: a dynamic index for multi-dimensional objects. In P. M. Stocker and W. Kent, editors, *Proceedings of the 13th International Conference on Very Large Databases (VLDB)*, pages 71–79, Brighton, United Kingdom, Sept. 1987. Also University of Maryland Computer Science Technical Report TR–1795, 1987.
85. K. Sevcik and N. Koudas. Filter trees for managing spatial data over a range of size granularities. In T. M. Vijayaraman, A. P. Buchmann, C. Mohan, and N. L. Sarda, editors, *Proceedings of the 22nd International Conference on Very Large Data Bases (VLDB)*, pages 16–27, Mumbai (Bombay), India, Sept. 1996.
86. C. A. Shaffer and H. Samet. Optimal quadtree construction algorithms. *Computer Vision, Graphics, and Image Processing*, 37(3):402–419, Mar. 1987.
87. C. A. Shaffer, H. Samet, and R. C. Nelson. QUILT: a geographic information system based on quadtrees. *International Journal of Geographical Information Systems*, 4(2):103–131, April–June 1990. Also University of Maryland Computer Science Technical Report TR–1885.1, July 1987.
88. R. Sivan and H. Samet. Algorithms for constructing quadtree surface maps. In *Proceedings of the 5th International Symposium on Spatial Data Handling*, volume 1, pages 361–370, Charleston, SC, Aug. 1992.
89. H.-W. Six and P. Widmayer. Spatial searching in geometric databases. In *Proceedings of the 4th IEEE International Conference on Data Engineering*, pages 496–503, Los Angeles, Feb. 1988.
90. J.-W. Song, K.-Y. Whang, Y.-K. Lee, M.-J. Lee, and S.-W. Kim. Spatial join processing using corner transformation. *IEEE Transactions on Knowledge and Data Engineering*, 11(4):688–695, July/August 1999.
91. M. Tamminen and H. Samet. Efficient octree conversion by connectivity labeling. *Computer Graphics*, 18(3):43–51, July 1984. Also in *Proceedings of the SIGGRAPH'84 Conference*, Minneapolis, MN, July 1984.
92. S. L. Tanimoto and T. Pavlidis. A hierarchical data structure for picture processing. *Computer Graphics and Image Processing*, 4(2):104–119, June 1975.
93. J. K. Uhlmann. Satisfying general proximity/similarity queries with metric trees. *Information Processing Letters*, 40(4):175–179, Nov. 1991.
94. T. Ulrich. Loose octrees. In M. A. DeLoura, editor, *Game Programming Gems*, pages 444–453. Charles River Media, Rockland, MA, 2000.
95. W. Wang, J. Yang, and R. Muntz. PK-tree: a spatial index structure for high dimensional point data. In K. Tanaka and S. Ghandeharizadeh, editors, *Proceedings of the 5th International Conference on Foundations of Data Organization and Algorithms (FODO)*, pages 27–36, Kobe, Japan, Nov. 1998. Also University of California at Los Angeles Computer Science Technical Report 980032, September 1998.
96. J. E. Warnock. A hidden line algorithm for halftone picture representation. Computer Science Technical Report TR 4–5, University of Utah, Salt Lake City, UT, May 1968.
97. J. E. Warnock. A hidden surface algorithm for computer generated half tone pictures. Computer Science Technical Report TR 4–15, University of Utah, Salt Lake City, UT, June 1969.
98. D. A. White and R. Jain. Similarity indexing with the SS-tree. In S. Y. W. Su, editor, *Proceedings of the 12th IEEE International Conference on Data Engineering*, pages 516–523, New Orleans, LA, Feb. 1996.
99. G. Wyvill and T. L. Kunii. A functional model for constructive solid geometry. *Visual Computer*, 1(1):3–14, July 1985.