HIERARCHICAL SPATIAL DATA STRUCTURES

Hanan Samet
Computer Science Department,
Center for Automation Research, and
Institute for Advanced Computer Studies
University of Maryland
College Park, Maryland 20742

Abstract

An overview is presented of the use of hierarchical spatial data structures such as the quadtree. They are based on the principle of recursive decomposition. The focus is on the representation of data used in image databases. The emphasis is on two-dimensional regions, points, rectangles, and lines.

1. INTRODUCTION

Hierarchical data structures are important representation techniques in the domains of computer vision, image processing, computer graphics, robotics, and geographic information systems. They are based on the principle of recursive decomposition (similar to *divide and conquer* methods). They are used primarily as devices to sort data of more than one dimension and different spatial types. The term *quadtree* is often used to describe this class of data structures. For a more extensive treatment of this subject, see [Same84a, Same88a, Same88b, Same88c, Same89a, Same89b].

Our presentation is organized as follows. Section 2 describes the region quadtree and presents an application for which it is well-suited. Section 3 briefly reviews the historical background of the origins of hierarchical data structures especially in the context of region data. Section 4 discusses a hierarchical representation of point data. Sections 4, 5, and 6 discuss hierarchical representations for point, rectangle, and line data, respectively, as well as give examples of their utility. Section 7 contains concluding remarks in the context of a geographic information system that makes use of these concepts.

2. REGION DATA

The term *quadtree* is used to describe a class of hierarchical data structures whose common property is that they are based on the principle of recursive decomposition of space. They can be differentiated on the following bases: (1) the type of data that they are used to represent, (2) the principle guiding the decomposition process, and (3) the resolution (variable or not). Currently, they are used for points, rectangles, regions, curves, surfaces, and volumes. The decomposition may be into equal parts on each level (termed a *regular decomposition*), or it may be governed by the input. The resolution of the decomposition (i.e., the number of times that the decomposition process is applied) may be fixed beforehand or it may be governed by properties of the input data.

The most common quadtree representation of data is the *region quadtree*. It is based on the successive subdivision of the image array into four equal-size quadrants. If the array does not consist entirely of 1s or entirely of 0s (i.e., the region does not cover the entire array), it is then subdivided into quadrants, subquadrants, etc., until blocks are obtained (possibly single pixels) that consist entirely of 1s or entirely of 0s. Thus, the region quadtree can be characterized as a variable resolution data structure.

As an example of the region quadtree, consider the region shown in Figure 1a which is represented by the $2^3 \times 2^3$ binary array in Figure 1b. Observe that the 1s correspond to picture elements (termed *pixels*) that are in the region and the 0s correspond to picture elements that are outside the region. The resulting blocks for the array of Figure 1b are shown in Figure 1c. This process is represented by a tree of degree 4.

In the tree representation, the root node corresponds to the entire array. Each son of a node represents a quadrant (labeled in order NW, NE, SW, SE) of the region represented by that node. The leaf nodes of the tree correspond to those blocks for

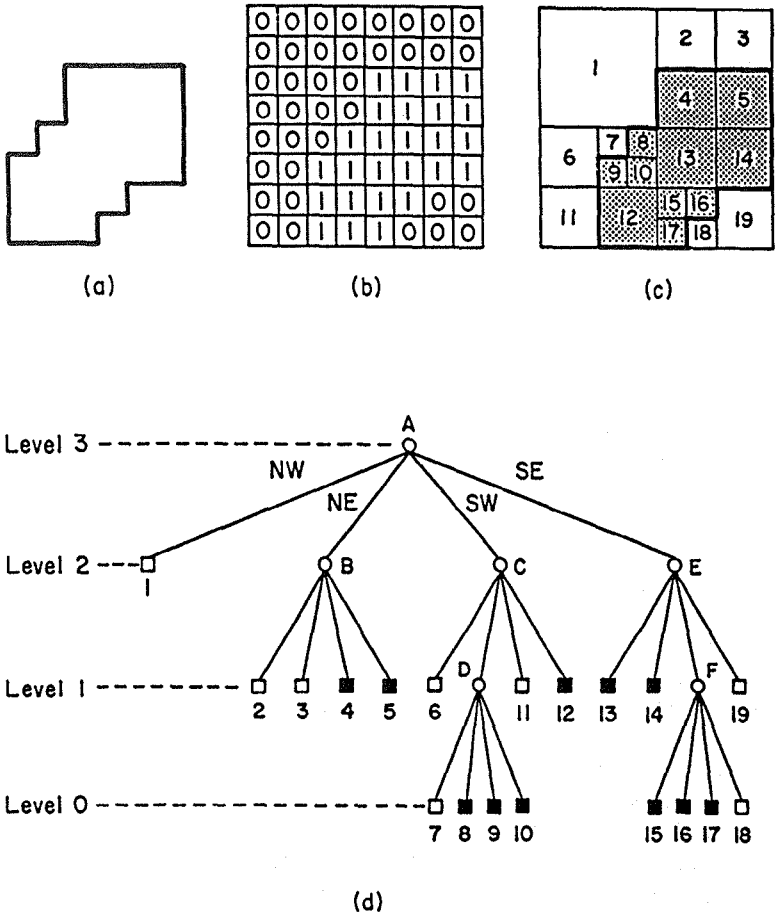

Figure 1. A region, its binary array, its maximal blocks, and the corresponding quadtree. (a) Region. (b) Binary array. (c) Block decomposition of the region in (a). Blocks in the region are shaded. (d) Quadtree representation of the blocks in (c).

which no further subdivision is necessary. A leaf node is said to be BLACK or WHITE, depending on whether its corresponding block is entirely inside or entirely outside of the represented region. All non-leaf nodes are said to be GRAY. The quadtree representation for Figure 1c is shown in Figure 1d. For an efficient algorithm to construct a quadtree from an image represented as a set of rows, see [Shaf87].

Quadtrees can also be used to represent non-binary images. In this case, we apply the same merging criteria to each color. For example, in the case of a landuse map, we simply merge all wheat growing regions, and likewise for corn, rice, etc. This is the approach taken by Samet *et al.* [Same84b].

For a binary image, set-theoretic operations such as union and intersection are quite simple to implement [Hunt78, Hunt79, Shne81a]. For example, the intersection of two region quadtrees yields a BLACK node only when the corresponding regions in both

quadtrees are BLACK. This operation is performed by simultaneously traversing three quadtrees. The first two trees correspond to the trees being intersected and the third tree represents the result of the operation. If any of the input nodes are WHITE, then the result is WHITE. When corresponding nodes in the input trees are GRAY, then their sons are recursively processed and a check is made for the mergibility of WHITE leaf nodes. The worst-case execution time of this algorithm is proportional to the sum of the number of nodes in the two input quadtrees. Note that as a result of actions (1) and (3), it is possible for the intersection algorithm to visit fewer nodes than the sum of the nodes in the two input quadtrees.

3. HISTORICAL BACKGROUND

Unfortunately, the term *quadtree* has taken on more than one meaning. The region quadtree, as described here, is a partition of space into a set of squares whose sides are all a power of two long. This formulation is due to Klinger [Klin71] who used the term Q-tree [Klin76], whereas Hunter [Hunt78] was the first to use the term quadtree in such a context. A similar partition of space into rectangular quadrants, also termed a quadtree, was used by Finkel and Bentley [Fink74]. It is an adaptation of the binary search tree to two dimensions (which can be easily extended to an arbitrary number of dimensions). It is primarily used to represent multidimensional point data. As an example, consider the point quadtree in Figure 2, which is built for the sequence Chicago, Mobile, Toronto, Buffalo, Denver, Omaha, Atlanta, and Miami. Note that its shape is highly dependent on the order in which the points are added to it.

The origin of the principle of recursive decomposition is difficult to ascertain. Below, in order to give some indication of the uses of the quadtree, we briefly trace some of its applications to image data. Morton [Mort66] used it as a means of indexing into a geographic database. Warnock [Warn69] implemented a hidden surface elimination algorithm using a recursive decomposition of the picture area. The picture area is

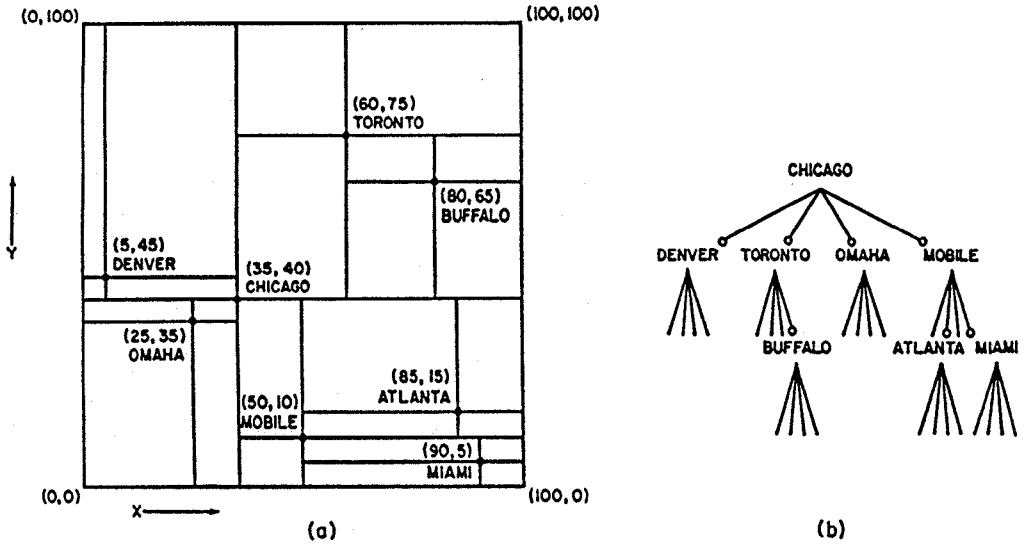

Figure 2. A point quadtree (b) and the records it represents (a).

repeatedly subdivided into successively smaller rectangles while searching for areas sufficiently simple to be displayed. Horowitz and Pavlidis [Horo76] used the quadtree as an initial step in a "split and merge" image segmentation algorithm.

The pyramid of Tanimoto and Pavlidis [Tani75] is a close relative of the region quadtree. It is a multiresolution representation which is is an exponentially tapering stack of arrays, each one-quarter the size of the previous array. It has been applied to the problems of feature detection and segmentation. In contrast, the region quadtree is a variable resolution data structure.

Quadtree-like data structures can also be used to represent images in three dimensions and higher. The octree [Hunt78, Jack80, Meag82, Redd78] data structure is the three-dimensional analog of the quadtree. It is constructed in the following manner. We start with an image in the form of a cubical volume and recursively subdivide it into eight congruent disjoint cubes (called octants) until blocks are obtained of a uniform color or a predetermined level of decomposition is reached. Figure 3a is an example of a simple three-dimensional object whose raster octree block decomposition is given in Figure 3b and whose tree representation is given in Figure 3c.

One of the motivations for the development of hierarchical data structures such as the quadtree is a desire to save space. The original formulation of the quadtree encodes it as a tree structure that uses pointers. This requires additional overhead to encode the internal nodes of the tree. In order to further reduce the space requirements, two other approaches have been proposed. The first treats the image as a collection of leaf nodes where each leaf is encoded by a base 4 number termed a *locational code*, corresponding to a sequence of directional codes that locate the leaf along a path from the root of the quadtree. It is analogous to taking the binary representation of the x and y coordinates of a designated pixel in the block (e.g., the one at the lower left corner) and interleaving them (i.e., alternating the bits for each coordinate). The encoding is also quite similar to a hashing function. It is difficult to determine the origin of this method (e.g., [Abel83, Garg82, Klin79, Mort66]).

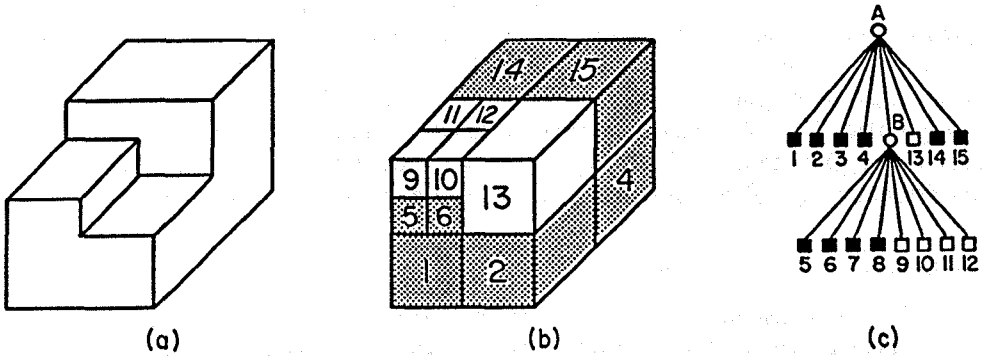

Figure 3. (a) Example three-dimensional object; (b) its octree block decomposition; and (c) its tree representation.

The second, termed a *DF-expression*, represents the image in the form of a traversal of the nodes of its quadtree [Kawa80]. It is very compact as each node type can be encoded with two bits. However, it is not easy to use when random access to nodes is desired. Samet and Webber [Same89c] show that for a static collection of nodes, an efficient implementation of the pointer-based representation is often more economical spacewise than a locational code representation. This is especially true for images of higher dimension.

Nevertheless, depending on the particular implementation of the quadtree we may not necessarily save space (e.g., in many cases a binary array representation may still be more economical than a quadtree). However, the effects of the underlying hierarchical aggregation on the execution time of the algorithms are more important. Most quadtree algorithms are simply preorder traversals of the quadtree and, thus, their execution time is generally a linear function of the number of nodes in the quadtree. A key to the analysis of the execution time of quadtree algorithms is the *Quadtree Complexity Theorem* [Hunt78, Hunt79] which states that:

> For a quadtree of depth q representing an image space of $2^q \times 2^q$ pixels where these pixels represent a region whose perimeter measured in pixel-widths is p, then the number of nodes in the quadtree cannot exceed $16 \cdot q - 11 + 16 \cdot p$.

Since under all but the most pathological cases (e.g., a small square of unit width centered in a large image), the region perimeter exceeds the base 2 logarithm of the width of the image containing the region, the Quadtree Complexity Theorem means that the size of the quadtree representation of a region is linear in the perimeter of the region.

The Quadtree Complexity Theorem holds for three-dimensional data [Meag80] where perimeter is replaced by surface area, as well as higher dimensions for which it is stated as follows.

> The size of the k-dimensional quadtree of a set of k-dimensional objects is proportional to the sum of the resolution and the size of the $(k-1)$-dimensional interfaces between these objects.

The Quadtree Complexity Theorem also directly impacts the analysis of the execution time of algorithms. In particular, most algorithms that execute on a quadtree representation of an image instead of an array representation have an execution time that is proportional to the number of blocks in the image rather than the number of pixels. In its most general case, this means that the application of a quadtree algorithm to a problem in d-dimensional space executes in time proportional to the analogous array-based algorithm in the $(d-1)$-dimensional space of the surface of the original d-dimensional image. Therefore, quadtrees act like dimension-reducing devices.

4. POINT DATA

Multidimensional point data can be represented in a variety of ways. The representation ultimately chosen for a specific task will be heavily influenced by the type of operations to be performed on the data. Our focus is on dynamic files (i.e., the number of data can grow and shrink at will) and on applications involving search. In Section 2 we briefly mentioned the point quadtree of Finkel and Bentley [Fink74] and showed its use. In this section we discuss the PR quadtree (P for point and R for region)

[Oren82, Same84a]. It is an adaptation of the region quadtree to point data which associates data points (that need not be discrete) with quadrants. The PR quadtree is organized in the same way as the region quadtree. The difference is that leaf nodes are either empty (i.e., WHITE) or contain a data point (i.e., BLACK) and its coordinates. A quadrant contains at most one data point. For example, Figure 4 is the PR quadtree corresponding to the data of Figure 2.

Data points are inserted into PR quadtrees in a manner analogous to that used to insert in a point quadtree - i.e., a search is made for them. Actually, the search is for the quadrant in which the data point, say A, belongs (i.e., a leaf node). If the quadrant is already occupied by another data point with different x and y coordinates, say B, then the quadrant must repeatedly be subdivided (termed *splitting*) until nodes A and B no longer occupy the same quadrant. This may result in many subdivisions, especially if the Euclidean distance between A and B is very small. The shape of the resulting PR quadtree is independent of the order in which data points are inserted into it. Deletion of nodes is more complex and may require collapsing of nodes - i.e., the direct counterpart of the node splitting process outlined above.

PR quadtrees, as well as other quadtree-like representations for point data, are especially attractive in applications that involve search. A typical query is one that requests the determination of all records within a specified distance of a given record - i.e., all cities within 100 miles of Washington, DC. The efficiency of the PR quadtree lies in its role as a pruning device on the amount of search that is required. Thus many records will not need to be examined. For example, suppose that in the hypothetical database of Figure 2 we wish to find all cities within 8 units of a data point with coordinates (84,10). In such a case, there is no need to search the NW, NE, and SW quadrants of the root (i.e., (50,50)). Thus we can restrict our search to the SE quadrant of the tree rooted at root. Similarly, there is no need to search the NW, NE, and SW

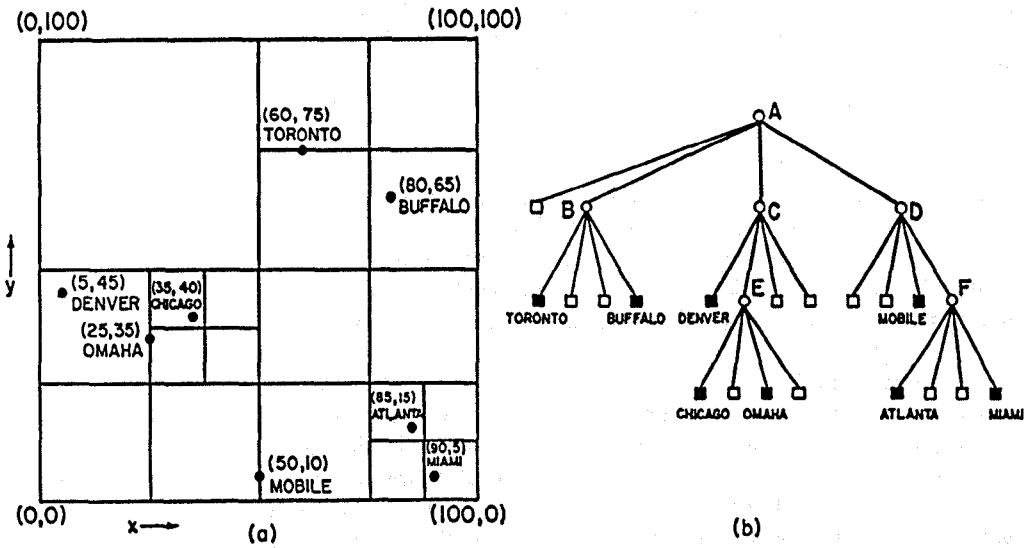

Figure 4. A PR quadtree (b) and the records it represents (a).

quadrants of the tree rooted at the SE quadrant (i.e., (75,25)). Note that the search ranges are usually orthogonally defined regions such as rectangles, boxes, etc. Other shapes are also feasible as the above example demonstrated (i.e., a circle).

5. RECTANGLE DATA

The rectangle data type lies somewhere between the point and region data types. Rectangles are often used to approximate other objects in an image for which they serve as the minimum rectilinear enclosing object. For example, bounding rectangles can be used in cartographic applications to approximate objects such as lakes, forests, hills, etc. [Mats84]. In such a case, the approximation gives an indication of the existence of an object. Of course, the exact boundaries of the object are also stored; but they are only accessed if greater precision is needed. For such applications, the number of elements in the collection is usually small, and most often the sizes of the rectangles are of the same order of magnitude as the space from which they are drawn.

Rectangles are also used in VLSI design rule checking as a model of chip components for the analysis of their proper placement. Again, the rectangles serve as minimum enclosing objects. In this application, the size of the collection is quite large (e.g., millions of components) and the sizes of the rectangles are several orders of magnitude smaller than the space from which they are drawn. Regardless of the application, the representation of rectangles involves two principal issues [Same88a]. The first is how to represent the individual rectangles and the second is how to organize the collection of the rectangles.

The representation that is used depends heavily on the problem environment. If the environment is static, then frequently the solutions are based on the use of the plane-sweep paradigm [Prep85], which usually yields optimal solutions in time and space. However, the addition of a single object to the database forces the re-execution of the algorithm on the entire database. We are primarily interested in dynamic problem environments. The data structures that are chosen for the collection of the rectangles are differentiated by the way in which each rectangle is represented.

One representation reduces each rectangle to a point in a higher dimensional space, and then treats the problem as if we have a collection of points. This is the approach of Hinrichs and Nievergelt [Hinr83, Hinr85]. Each rectangle is a Cartesian product of two one-dimensional intervals where each interval is represented by its centroid and extent. The collection of rectangles is, in turn, represented by a grid file [Niev84], which is a hierarchical data structure for points.

The second representation is region-based in the sense that the subdivision of the space from which the rectangles are drawn depends on the physical extent of the rectangle - not just one point. Representing the collection of rectangles, in turn, with a tree-like data structure has the advantage that there is a relation between the depth of node in the tree and the size of the rectangle(s) that are associated with it. Interestingly, some of the region-based solutions make use of the same data structures that are used in the solutions based on the plane-sweep paradigm. In the remainder of this section, we give an example of a pair of region-based representations.

The *MX-CIF quadtree* of Kedem [Kede81] (see also Abel and Smith [Abel83]) is a region-based representation where each rectangle is associated with the quadtree node corresponding to the smallest block which contains it in its entirety. Subdivision ceases whenever a node's block contains no rectangles. Alternatively, subdivision can also cease once a quadtree block is smaller than a predetermined threshold size. This threshold is often chosen to be equal to the expected size of the rectangle [Kede81]. For example, Figure 5 is the MX-CIF quadtree for a collection of rectangles. Note that rectangle F occupies an entire block and hence it is associated with the block's father. Also rectangles can be associated with both terminal and non-terminal nodes.

It should be clear that more than one rectangle can be associated with a given enclosing block and, thus, often we find it useful to be able to differentiate between them. Kedem proposes to do so in the following manner. Let P be a quadtree node with centroid (CX,CY), and let S be the set of rectangles that are associated with P. Members of S are organized into two sets according to their intersection (or collinearity of their sides) with the lines passing through the centroid of P 's block - i.e., all members of S that intersect the line $x=CX$ form one set and all members of S that intersect the line $y=CY$ form the other set.

If a rectangle intersects both lines (i.e., it contains the centroid of P 's block), then we adopt the convention that it is stored with the set associated with the line through $x=CX$. These subsets are implemented as binary trees (really tries), which in actuality are one-dimensional analogs of the MX-CIF quadtree. For example, Figure 6 illustrates the binary tree associated with the y axes passing through the root and the NE son of the root of the MX-CIF quadtree of Figure 5. Interestingly, the MX-CIF quadtree is a two-dimensional analog of the interval tree [Edel80, McCr80], which is a data structure that is used to support optimal solutions based on the plane-sweep paradigm to some rectangle problems.

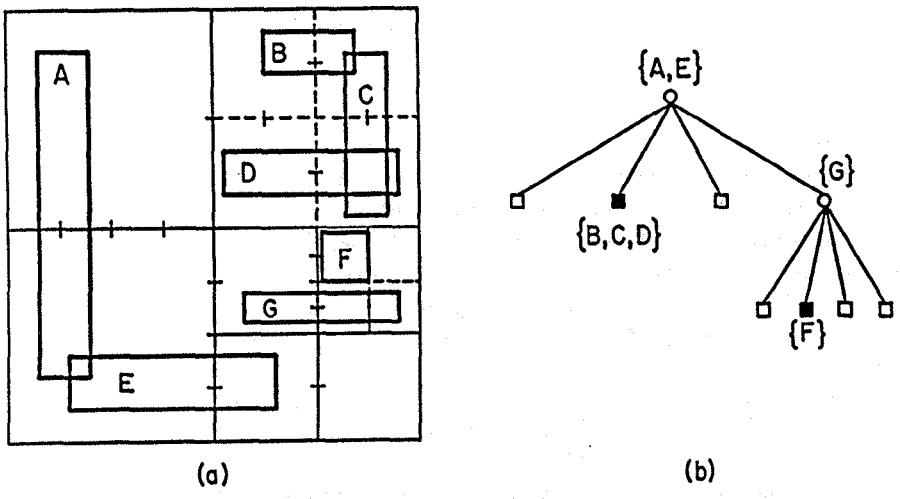

Figure 5. MX-CIF quadtree. (a) Collection of rectangles and the block decomposition induced by the MX-CIF quadtree. (b) The tree representation of (a).

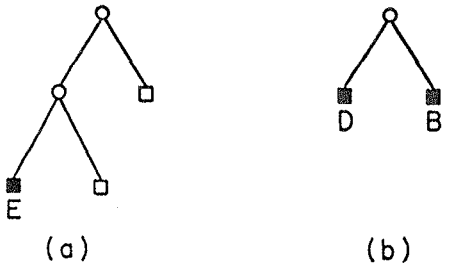

Figure 6. Binary trees for the y axes passing through (a) the root of the MX-CIF quadtree in Figure 6 and (b) the NE son of the root of the MX-CIF quadtree in Figure 6.

The R-tree [Gutt84] is a hierarchical data structure that is derived from the B-tree [Come79]. Each node in the tree is a d-dimensional rectangle corresponding to the smallest rectangle that encloses its son nodes which are also d-dimensional rectangles. The leaf nodes are the actual rectangles in the database. Often, the nodes correspond to disk pages and, thus, the parameters defining the tree are chosen so that a small number of nodes is visited during a spatial query. Note that rectangles corresponding to different nodes may overlap.

Also, a rectangle may be spatially contained in several nodes, yet it can only be associated with one node. This means that a spatial query may often require several nodes to be visited before ascertaining the presence or absence of a particular rectangle. This problem can be alleviated by using the R^+-tree [Falo87, Sell87] for which all bounding rectangles (i.e., at levels other than the leaf) are non-overlapping. This means that a given rectangle will often be associated with several bounding rectangles. In this case, retrieval time is sped up at the cost of an increase in the height of the tree. Note that B-tree performance guarantees are not valid for the R^+-tree - i.e., pages are not guaranteed to be 50% full without very complicated record update procedures.

6. LINE DATA

Sections 2 and 3 were devoted to the region quadtree, an approach to region representation that is based on a description of the region's interior. In this section, we focus on a representation that specifies the boundaries of regions. This is done in the more general context of data structures for curvilinear data. The simplest representation is the polygon in the form of vectors which are usually specified in the form of lists of pairs of x and y coordinate values corresponding to their start and end points. One of the most common representations is the chain code [Free74] which is an approximation of a polygon. There has also been a considerable amount of interest recently in hierarchical representations. These are primarily based on rectangular approximations to the data as well as on a regular decomposition in two dimensions.

The *strip tree* [Ball81] is a hierarchical representation of a single curve that is obtained by successively approximating segments of it by enclosing rectangles. The data structure consists of a binary tree whose root represents the bounding rectangle of the entire curve. For example, consider Figure 7a where the curve between points P and Q, at locations (x_P,y_P) and (x_Q,y_Q) respectively, is modeled by a strip tree. The rectangle

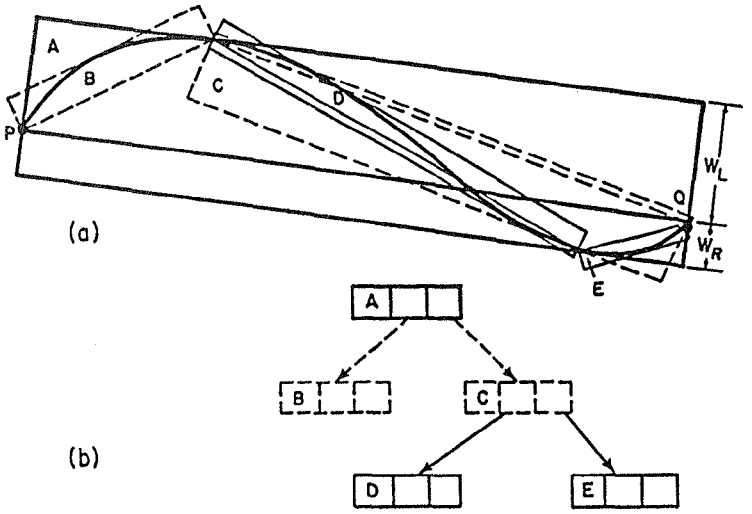

Figure 7. A curve between points P and Q. (a) Its decomposition into strips; and (b) the corresponding strip tree.

associated with the root, A in this example, corresponds to a rectangular strip, that encloses the curve, whose sides are parallel to the line joining the endpoints of the curve (i.e., P and Q). The curve is then partitioned in two at one of the locations where it touches the bounding rectangle. Each subcurve is then surrounded by a bounding rectangle and the partitioning process is applied recursively. This process stops when the width of each strip is less than a predetermined value. The strip tree is implemented as a binary tree (Figure 7b) where each node contains eight fields. Four fields contain the x and y coordinates of the endpoints, two fields contain pointers to the two sons of the node, and two fields contain information about the width of the strip (i.e., W_L and W_R in Figure 7a).

Figure 7 is a relatively simple example. In order to be able to cope with more complex curves, the notion of a strip tree must be extended. In particular, closed curves and curves that extend past their endpoints require some special treatment. The general idea is that these curves are enclosed by rectangles which are split into two rectangular strips and from now on the strip tree is used as before. For a related approach that does not require these extensions, see the arc tree of Günther [Günt87]. Its subdivision rule consists of a regular decomposition of a curve based on its length.

Like point and region quadtrees, strip trees are useful in applications that involve search and set operations. For example, suppose we wish to determine whether a road crosses a river. Using a strip tree representation for these features, answering this query means basically performing an intersection of the corresponding strip trees. Three cases are possible as is shown in Figure 8. Figures 8a and 8b correspond to the answers NO and YES respectively while Figure 8c requires us to descend further down the strip tree. Notice the distinction between the task of detecting the possibility of an intersection and the task of computing the actual intersection, if one exists. The strip tree is well suited to the former task. Other operations that can be performed efficiently by using the strip tree data structure include the computation of the length of a curve, areas of closed

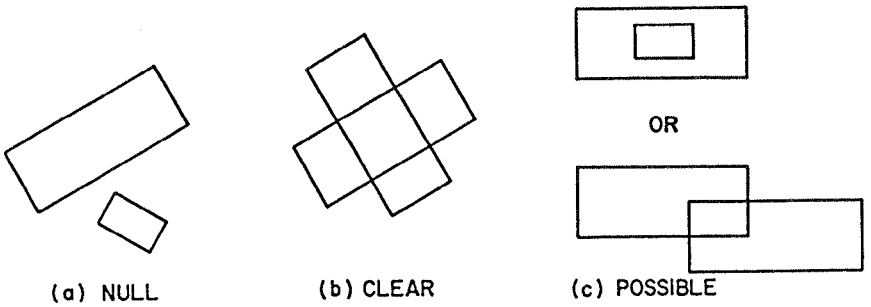

Figure 8. Three possible results of intersecting two strip trees. (a) Null. (b) Clear. (c) Possible.

curves, intersection of curves with areas, point membership, etc.

The strip tree is similar to the point quadtree in the sense that the points at which the curve is decomposed depend on the data. In contrast, a region quadtree approach has fixed decomposition points. Similarly, strip tree methods approximate curvilinear data with rectangles while methods based on the region-quadtree achieve analogous results by use of a collection of disjoint squares having sides of length power of two. In the following we discuss a number of adaptations of the region quadtree for representing curvilinear data.

The *edge quadtree* [Shne81b, Warn69] is an attempt to store linear feature information (e.g., curves) for an image (binary and gray-scale) in a manner analogous to that used for storing region information. A region containing a linear feature or part thereof is subdivided into four squares repeatedly until a square is obtained that contains a single curve that can be approximated by a single straight line (e.g., Figure 9 where the maximum level of decomposition is 4). Each leaf node contains the following

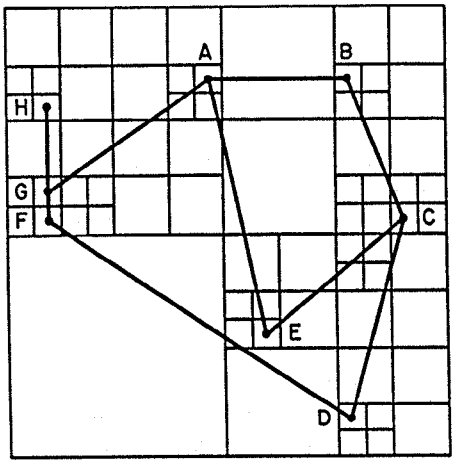

Figure 9. An edge quadtree.

information about the edge passing through it: magnitude (i.e., 1 in the case of a binary image or the intensity in case it is a gray-scale image), direction, intercept, and a directional error term (i.e., the error induced by approximating the curve by a straight line using a measure such as least squares). If an edge terminates within a node, then a special flag is set and the intercept denotes the point at which the edge terminates. Applying this process leads to quadtrees in which long edges are represented by large leaves or a sequence of large leaves. However, small leaves are required in the vicinity of corners or intersecting edges. Of course, many leaves will contain no edge information at all.

The PM quadtree family [Same85, Nels86] (see also edge-EXCELL [Tamm81]) represents an attempt to overcome some of the problems associated with the edge quadtree in the representation of collections of polygons (termed *polygonal maps*). In particular, the edge quadtree is an approximation because vertices are represented by pixels. Moreover, it is difficult to detect the presence of a vertex when more than five line segments meet. There are a number of variants of the PM quadtree. These variants are either vertex-based or edge-based. They are all built by applying the principle of repeatedly breaking up the collection of vertices and edges (forming the polygonal map) until obtaining a subset that is sufficiently simple so that it can be organized by some other data structure.

The PM quadtrees of Samet and Webber [Same85] are vertex-based. We illustrate the PM_1 quadtree. It is based on a decomposition rule stipulating that partitioning occurs as long as a block contains more than one line segment unless the line segments are all incident at the same vertex which is also in the same block (e.g., Figure 10).

Samet, Shaffer, and Webber [Same87] show how to compute the maximum depth of the PM_1 quadtree for a polygonal map in a limited, but typical, environment. They

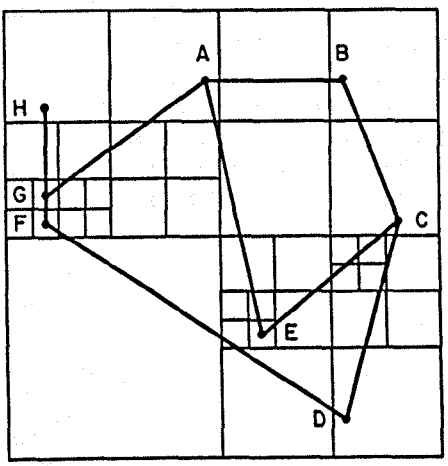

Figure 10. Example PM_1 quadtree.

consider a polygonal map whose vertices are drawn from a grid (say $2^n \times 2^n$), and do not permit edges to intersect at points other than the grid points (i.e., vertices). In such a case, the depth of any leaf node is bounded from above by $4n+1$. This enables a determination of the maximum amount of storage that will be necessary for each node.

A similar representation has been devised for three-dimensional images [Ayal85, Carl85, Fuji85, Hunt81, Nava86, Quin82, Tamm81, Vand84]. The decomposition criteria are such that no node contains more than one face, edge, or vertex unless the faces all meet at the same vertex or are adjacent to the same edge. For example, Figure 11b is a PM_1 octree decomposition of the object in Figure 11a. This representation is quite useful since its space requirements for polyhedral objects are significantly smaller than those of a conventional octree.

The PMR quadtree [Nels86] is an edge-based variant of the PM quadtree (see also edge-EXCELL [Tamm81]). It makes use of a probabilistic splitting rule. A node is permitted to contain a variable number of line segments. A line segment is stored in a PMR quadtree by inserting it into the nodes corresponding to all the blocks that it intersects. During this process, the occupancy of each node that is intersected by the line segment is checked to see if the insertion causes it to exceed a predetermined *splitting threshold.* If the splitting threshold is exceeded, then the node's block is split *once*, and only once, into four equal quadrants.

On the other hand, a line segment is deleted from a PMR quadtree by removing it from the nodes corresponding to all the blocks that it intersects. During this process, the occupancy of the node and its siblings is checked to see if the deletion causes the total number of line segments in them to be less than the predetermined splitting threshold. If the splitting threshold exceeds the occupancy of the node and its siblings, then they are merged and the merging process is reapplied to the resulting node and its siblings. Notice the asymmetry between the splitting and merging rules.

Members of the PM quadtree family can be easily adapted to deal with fragments that result from set operations such as union and intersection so that there is no data degradation when fragments of line segments are subsequently recombined. Their use yields an exact representation of the lines - not an approximation. To see how this is achieved, let us define a *q-edge* to be a segment of an edge of the original polygonal map that either spans an entire block in the PM quadtree or extends from a boundary of a block to a vertex within the block (i.e., when the block contains a vertex).

Figure 11. (a) Example three-dimensional object; and (b) its corresponding PM_1 octree.

Each q-edge is represented by a pointer to a record containing the endpoints of the edge of the polygonal map of which the q-edge is a part [Nels86]. The line segment descriptor stored in a node only implies the presence of the corresponding q-edge - it does not mean that the entire line segment is present as a lineal feature. The result is a consistent representation of line fragments since they are stored exactly and, thus, they can be deleted and reinserted without worrying about errors arising from the roundoffs induced by approximating their intersection with the borders of the blocks through which they pass.

7. CONCLUDING REMARKS

The use of hierarchical data structures in image databases enables the focussing of computational resources on the interesting subsets of data. Thus, there is no need to expend work where the payoff is small. Although many of the operations for which they are used can often be performed equally as efficiently, or more so, with other data structures, hierarchical data structures are attractive because of their conceptual clarity and ease of implementation.

When the hierarchical data structures are based on the principle of regular decomposition, we have the added benefit of a spatial index. All features, be they regions, points, rectangles, lines, volumes, etc., can be represented by maps which are in registration. In fact, such a system has been built [Same84b] for representing geographic information. In this case, the quadtree is implemented as a collection of leaf nodes where each leaf node is represented by its locational code. The collection is in turn represented as a B-tree [Come79]. There are leaf nodes corresponding to region, point, and line data.

The disadvantage of quadtree methods is that they are shift sensitive in the sense that their space requirements are dependent on the position of the origin. However, for complicated images the optimal positioning of the origin will usually lead to little improvement in the space requirements. The process of obtaining this optimal positioning is computationally expensive and is usually not worth the effort [Li82].

The fact that we are working in a digitized space may also lead to problems. For example, the rotation operation is not generally invertible. In particular, a rotated square usually cannot be represented accurately by a collection of rectilinear squares. However, when we rotate by 90°, then the rotation is invertible. This problem arises whenever one uses a digitized representation. Thus, it is also common to the array representation.

ACKNOWLEDGMENTS

This work was supported by the National Science Foundation under Grant IRI-8802457. I would like to acknowledge the many valuable discussions that I have had with Michael B. Dillencourt, Randal C. Nelson, Azriel Rosenfeld, Clifford A. Shaffer, Markku Tamminen, and Robert E. Webber.

REFERENCES

1. [Abel83] - 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(October 1983), 1-13.

2. [Ayal85] - 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(January 1985), 41-59.

3. [Ball81] - D.H. Ballard, Strip trees: A hierarchical representation for curves, *Communications of the ACM 24*, 5(May 1981), 310-321 (see also corrigendum, *Communications of the ACM 25*, 3(March 1982), 213).

4. [Carl85] - I. Carlbom, I. Chakravarty, and D. Vanderschel, A hierarchical data structure for representing the spatial decomposition of 3-D objects, *IEEE Computer Graphics and Applications 5*, 4(April 1985), 24-31.

5. [Come79] - D. Comer, The Ubiquitous B-tree, *ACM Computing Surveys 11*, 2(June 1979), 121-137.

6. [Edel80] - H. Edelsbrunner, Dynamic rectangle intersection searching, Institute for Information Processing Report 47, Technical University of Graz, Graz, Austria, February 1980.

7. [Falo87] - C. Faloutsos, T. Sellis, and N. Roussopoulos, Analysis of object oriented spatial access methods, *Proceedings of the SIGMOD Conference*, San Francisco, May 1987, 426-439.

8. [Fink74] - R.A. Finkel and J.L. Bentley, Quad trees: a data structure for retrieval on composite keys, *Acta Informatica 4*, 1(1974), 1-9.

9. [Free74] - H. Freeman, Computer processing of line-drawing images, *ACM Computing Surveys 6*, 1(March 1974), 57-97.

10. [Fuji85] - K. Fujimura and T.L. Kunii, A hierarchical space indexing method, *Proceedings of Computer Graphics'85*, Tokyo, 1985, T1-4, 1-14.

11. [Garg82] - I. Gargantini, An effective way to represent quadtrees, *Communications of the ACM 25*, 12(December 1982), 905-910.

12. [Günt87] - O. Günther, Efficient structures for geometric data management, Ph.D. dissertation, UCB/ERL M87/77, Electronics Research Laboratory, College of Engineering, University of California at Berkeley, Berkeley, CA, 1987 (Lecture Notes in Computer Science 337, Springer-Verlag, Berlin, 1988).

13. [Gutt84] - A. Guttman, R-trees: a dynamic index structure for spatial searching, *Proceedings of the SIGMOD Conference*, Boston, June 1984, 47-57.

14. [Hinr85] - K. Hinrichs, The grid file system: implementation and case studies of applications, Ph.D. dissertation, Institut fur Informatik, ETH, Zurich, Switzerland, 1985.

15. [Hinr83] - K. Hinrichs and J. Nievergelt, The grid file: a data structure designed to support proximity queries on spatial objects, *Proceedings of the WG'83 (International Workshop on Graphtheoretic Concepts in Computer Science)*, M. Nagl and J. Perl, Eds., Trauner Verlag, Linz, Austria, 1983, 100-113.

16. [Horo76] - S.L. Horowitz and T. Pavlidis, Picture segmentation by a tree traversal algorithm, *Journal of the ACM 23*, 2(April 1976), 368-388.

17. [Hunt78] - G.M. Hunter, Efficient computation and data structures for graphics, Ph.D. dissertation, Department of Electrical Engineering and Computer Science, Princeton University, Princeton, NJ, 1978.

18. [Hunt81] - G.M. Hunter, Geometrees for interactive visualization of geology: an evaluation, System Science Department, Schlumberger-Doll Research, Ridgefield, CT, 1981.

19. [Hunt79] - G.M. Hunter and K. Steiglitz, Operations on images using quad trees, *IEEE Transactions on Pattern Analysis and Machine Intelligence 1*, 2(April 1979), 145-153.

20. [Jack80] - C.L. Jackins and S.L. Tanimoto, Oct-trees and their use in representing three-dimensional objects, *Computer Graphics and Image Processing 14*, 3(November 1980), 249-270.

21. [Kawa80] - E. Kawaguchi and T. Endo, On a method of binary picture representation and its application to data compression, *IEEE Transactions on Pattern Analysis and Machine Intelligence 2*, 1(January 1980), 27-35.

22. [Kede81] - G. Kedem, The Quad-CIF tree: a data structure for hierarchical on-line algorithms, *Proceedings of the Nineteenth Design Automation Conference*, Las Vegas, June 1982, 352-357.

23. [Klin71] - A. Klinger, Patterns and Search Statistics, in *Optimizing Methods in Statistics*, J.S. Rustagi, Ed., Academic Press, New York, 1971, 303-337.

24. [Klin76] - A. Klinger and C.R. Dyer, Experiments in picture representation using regular decomposition, *Computer Graphics and Image Processing 5*, 1(March 1976), 68-105.

25. [Klin79] - A. Klinger and M.L. Rhodes, Organization and access of image data by areas, *IEEE Transactions on Pattern Analysis and Machine Intelligence 1*, 1(January 1979), 50-60.

26. [Li82] - M. Li, W.I. Grosky, and R. Jain, Normalized quadtrees with respect to translations, *Computer Graphics and Image Processing 20*, 1(September 1982), 72-81.

27. [Mats84] - T. Matsuyama, L.V. Hao, and M. Nagao, A file organization for geographic information systems based on spatial proximity, *Computer Vision, Graphics, and Image Processing 26*, 3(June 1984), 303-318.

28. [McCr80] - E.M. McCreight, Efficient algorithms for enumerating intersecting intervals and rectangles, Xerox Palo Alto Research Center Report CSL-80-09, Palo Alto, California, June 1980.

29. [Meag80] - D. Meagher, Octree encoding: a new technique for the representation, The manipulation, and display of arbitrary 3-d objects by computer, Technical Report IPL-TR-80-111, Image Processing Laboratory, Rensselaer Polytechnic Institute, Troy, New York, October 1980.

30. [Meag82] - D. Meagher, Geometric modeling using octree encoding, *Computer Graphics and Image Processing 19*, 2(June 1982), 129-147.

31. [Mort66] - G.M. Morton, A computer oriented geodetic data base and a new technique in file sequencing, IBM Ltd., Ottawa, Canada, 1966.

32. [Nava86] - I. Navazo, Contribució a les tècniques de modelat geomètric d'objectes polièdrics usant la codificació amb arbres octals, Ph.D. dissertation, Escola Tecnica Superior d'Enginyers Industrials, Department de Metodes Informatics, Universitat Politechnica de Barcelona, Barcelona, Spain, January 1986.

33. [Nels86] - R.C. Nelson and H. Samet, A consistent hierarchical representation for vector data, *Computer Graphics 20*, 4(August 1986), pp. 197-206 (also *Proceedings of the SIGGRAPH'86 Conference*, Dallas, August 1986).

34. [Niev84] - J. Nievergelt, H. Hinterberger, and K.C. Sevcik, The grid file: an adaptable, symmetric multikey file structure, *ACM Transactions on Database Systems 9*, 1(March 1984), 38-71.

35. [Oren82] - J.A. Orenstein, Multidimensional tries used for associative searching, *Information Processing Letters 14*, 4(June 1982), 150-157.

36. [Prep85] - F.P. Preparata and M.I. Shamos, *Computational Geometry: An Introduction*, Springer-Verlag, New York, 1985.

37. [Quin82] - K.M. Quinlan and J.R. Woodwark, A spatially-segmented solids database - justification and design, *Proceedings of CAD 82 Conference*, Butterworth, Guildford, United Kingdom, 1982, 126-132.

38. [Redd78] - D.R. Reddy and S. Rubin, Representation of three-dimensional objects, CMU-CS-78-113, Computer Science Department, Carnegie-Mellon University, Pittsburgh, April 1978.

39. [Same84a] - H. Samet, The quadtree and related hierarchical data structures, *ACM Computing Surveys 16*, 2(June 1984), 187-260.

40. [Same88a] - H. Samet, Hierarchical representations of collections of small rectangles, *ACM Computing Surveys 20*, 4(December 1988), 271-309.

41. [Same89a] - H. Samet, *The Design and Analysis of Spatial Data Structures*, Addison-Wesley, Reading, MA, 1989.

42. [Same89b] - H. Samet, *Applications of Spatial Data Structures: Computer Graphics, Image Processing and GIS*, Addison-Wesley, Reading, MA, 1989.

43. [Same84b] - H. Samet, A. Rosenfeld, C.A. Shaffer, and R.E. Webber, A geographic information system using quadtrees, *Pattern Recognition 17*, 6 (November/December 1984), 647-656.

44. [Same87] - H. Samet, C.A. Shaffer, and R.E. Webber, Digitizing the plane with cells of non-uniform size, *Information Processing Letters 24*, 6(April 1987), 369-375.

45. [Same85] - H. Samet and R.E. Webber, Storing a collection of polygons using quadtrees, *ACM Transactions on Graphics 4*, 3(July 1985), 182-222 (also *Proceedings of Computer Vision and Pattern Recognition 83*, Washington, DC, June 1983, 127-132).

46. [Same89c] - H. Samet and R.E. Webber, A comparison of the space requirements of multi-dimensional quadtree-based file structures, to appear in *The Visual Computer* (also University of Maryland Computer Science TR-1711).

47. [Same88b] - H. Samet and R.E. Webber, Hierarchical data structures and algorithms for computer graphics. Part I. Fundamentals, *IEEE Computer Graphics and Applications 8*, 3(May 1988), 48-68.

48. [Same88c] - H. Samet and R.E. Webber, Hierarchical data structures and algorithms for computer graphics. Part II. Applications, *IEEE Computer Graphics and Applications 8*, 4(July 1988), 59-75.

49. [Sell87] - T. Sellis, N. Roussopoulos, and C. Faloutsos, The R^+-tree: a dynamic index for multi-dimensional objects, Computer Science TR-1795, University of Maryland, College Park, MD, February 1987.

50. [Shaf87] - C.A. Shaffer and H. Samet, Optimal quadtree construction algorithms, *Computer Vision, Graphics, and Image Processing 37*, 3(March 1987), 402-419.

51. [Shne81a] - M. Shneier, Calculations of geometric properties using quadtrees, *Computer Graphics and Image Processing 16*, 3(July 1981), 296-302.

52. [Shne81b] - M. Shneier, Two hierarchical linear feature representations: edge pyramids and edge quadtrees, *Computer Graphics and Image Processing 17*, 3(November 1981), 211-224.

53. [Tamm81] - M. Tamminen, The EXCELL method for efficient geometric access to data, *Acta Polytechnica Scandinavica*, Mathematics and Computer Science Series No. 34, Helsinki, 1981.

54. [Tani75] - S. Tanimoto and T. Pavlidis, A hierarchical data structure for picture processing, *Computer Graphics and Image Processing 4*, 2(June 1975), 104-119.

55. [Vand84] - D.J. Vanderschel, Divided leaf octal trees, Research Note, Schlumberger-Doll Research, Ridgefield, CT, March 1984.

56. [Warn69] - J.L. Warnock, A hidden surface algorithm for computer generated half tone pictures, Computer Science Department TR 4-15, University of Utah, Salt Lake City, June 1969.