HIERARCHICAL DATA STRUCTURES FOR IMAGE DATABASES*

Hanan Samet
Computer Science Department
University of Maryland
College Park, Maryland 20742

ABSTRACT

An overview of hierarchical data structures for representing images, such as the quadtree, is presented. They are based on the principle of recursive decomposition. The emphasis is on the representation of data used in applications in image processing, computer vision, computer graphics, robotics, and geographic information systems. There is a greater emphasis on region data (i.e., 2-dimensional shapes) and to a lesser extent on point, line, and 3-dimensional data.

1. INTRODUCTION

Hierarchical data structures are becoming increasingly 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). One such data structure is the quadtree. As we shall see, the term *quadtree* has taken on a generic meaning. In this overview it is our goal to show how a number of data structures used in different domains are related to each other and to quadtrees. Our presentation concentrates on these different representations and illustrates how some basic operations which use them are performed. For a more extensive treatment of this subject, including a comprehensive set of references, see [Same84a, Same87].

This overview is organized as follows. Section 2 discusses the historical background of the origins of hierarchical data structures. Section 3 shows how some key operations are performed on region data. Section 4 describes hierarchical representations for point data while Section 5 does the same for line data. Section 6 contains concluding remarks in the context of a geographic information system that makes use of these concepts.

*The support of the National Science Foundation under Grant DCR–86–05557 is gratefully acknowledged.

TH0166-9/87/0000-0387$01.00 © 1987 IEEE

2. BACKGROUND

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 point data, regions, curves, surfaces, and volumes. The decomposition may be into equal parts on each level (i.e., regular polygons and 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.

Our first example of quadtree representation of data is concerned with the representation of region data. The most studied quadtree approach to region representation, termed a *region quadtree*, is based on the successive subdivision of the image array into four equal-size quadrants. If the array does not consist entirely of 1's or entirely of 0's (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 1's or entirely of 0's; i.e., each block is entirely contained in the region or entirely disjoint from it. Thus the region quadtree can be characterized as a variable resolution data structure. As an example, consider the region shown in Figure 1a which is represented by the 2^3 by 2^3 binary array in Figure 1b. Observe that the 1's correspond to picture elements (termed *pixels*) that are in the region and the 0's 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 (i.e., each non-leaf node has four sons). 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 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.

The example described in Figure 1 corresponds to a binary image. The extension to non-binary images is straightforward. There are two choices. First, we can apply the same merging criteria to each color. For example, in the case of a landuse map that is represented as a region quadtree, we would merge all wheat growing regions, and likewise for corn, rice, etc. This is the approach taken by Samet *et al.* [Same84b]. In the case of gray-scale images we can take advantage of the fact that the image is continuous and thus adjacent image elements will usually have gray level values that are almost identical. For example, in a 256-level image, we would expect elements with codes 127 and 128 to be adjacent. In fact, this is the approach taken by Kawaguchi *et al.* [Kawa83] who use a sequence of m binary-valued quadtrees to encode image data of 2^m gray levels, where the various gray levels are encoded by use of Gray codes. This should lead to compaction (i.e., larger sized blocks), since the Gray code guarantees that adjacent gray level values differ by only one binary digit.

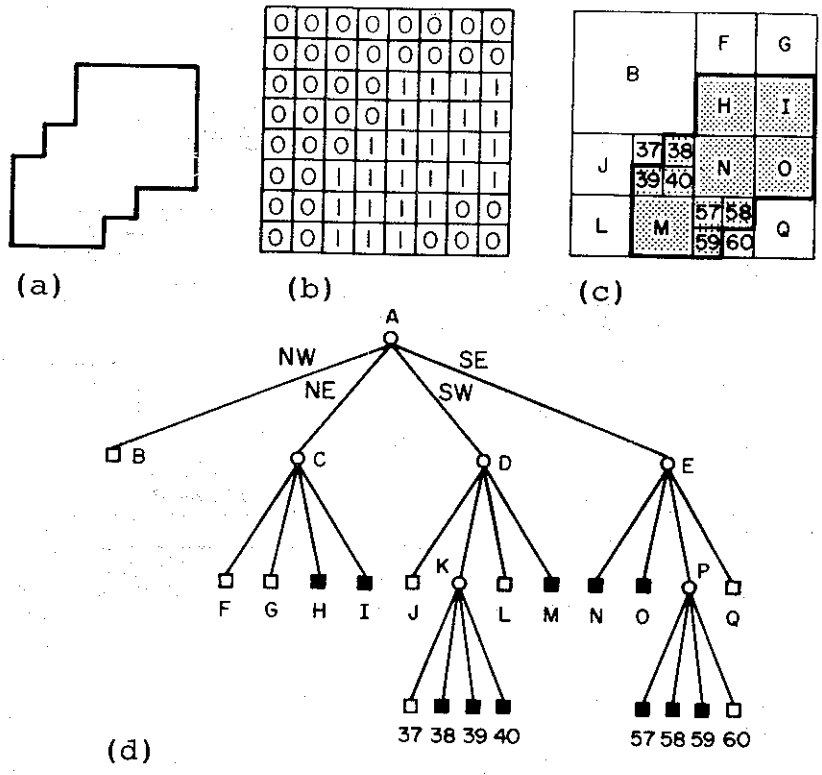

Figure 1. A region, its binary array, its maximal blocks, and the corresponding quadtree. (a) Region. (b) Binary array. (c) Block decomposition of the region in (a). Blocks in the region are shaded. (d) Quadtree representation of the blocks in (c).

Unfortunately, the term *quadtree* has taken on more than one meaning. The region quadtree, as shown above, 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, 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 and we shall refer to it as a *point quadtree* where confusion with a region quadtree is possible. 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, and incompletely, trace some of its applications to image data. Morton [Mort66] used it as a means of indexing into a geographic database. Warnock [Warn68] implemented a hidden surface elimination algorithm using a recursive decomposition of the picture area. The picture area is repeatedly subdivided into successively smaller rectangles while searching

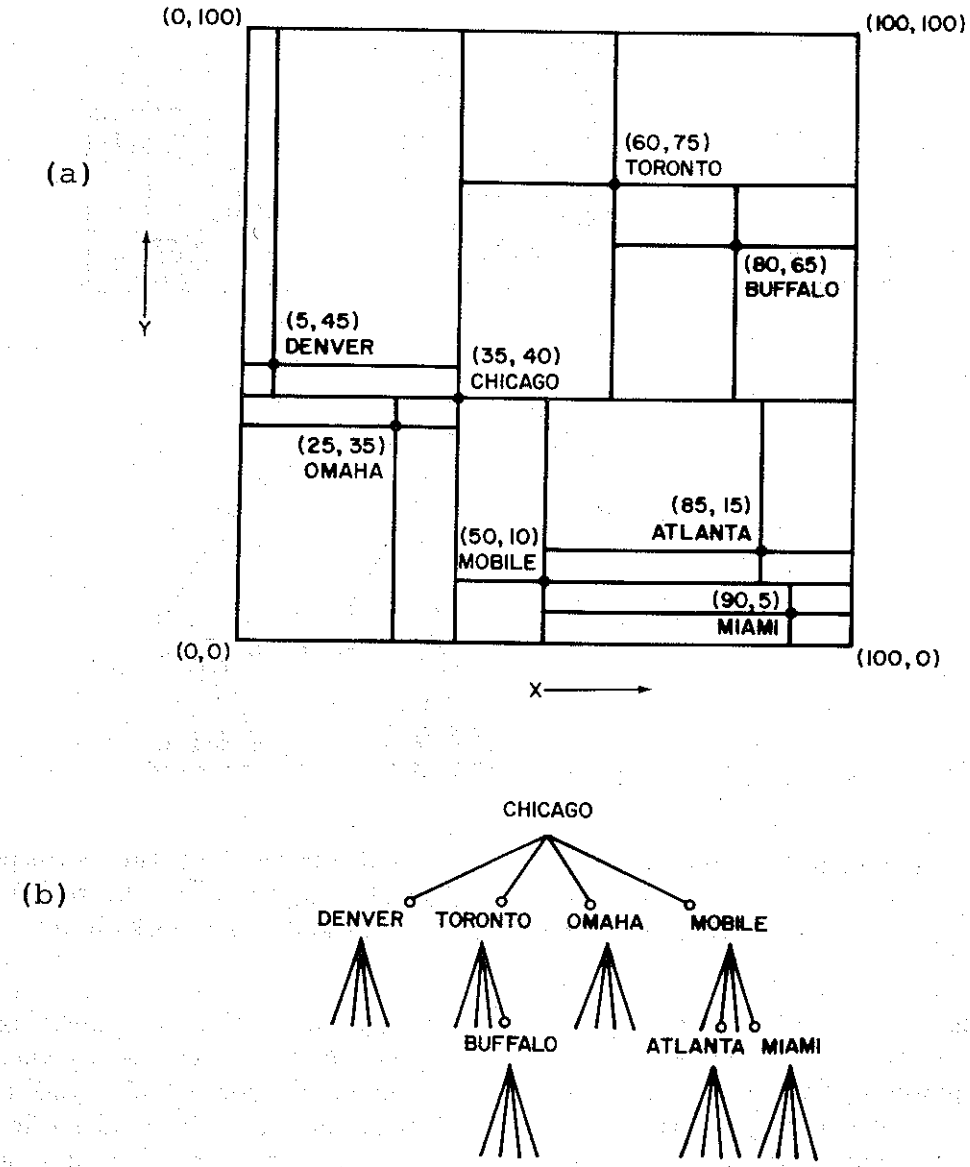

Figure 2. A point quadtree (b) and the records it represents (a).

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. A pyramid 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. For more details on pyramids, see the collection of papers edited by Rosenfeld [Rose83].

Quadtree-like data structures can also be used to represent images in three dimensions and higher. The octree [Hunt78, Meag82] 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 of a uniform color are obtained, or a predetermined level of decomposition is reached. Figure 3a is an example of a simple three-dimensional object whose octree block decomposition is given in Figure 3b and whose tree representation is given in Figure 3c.

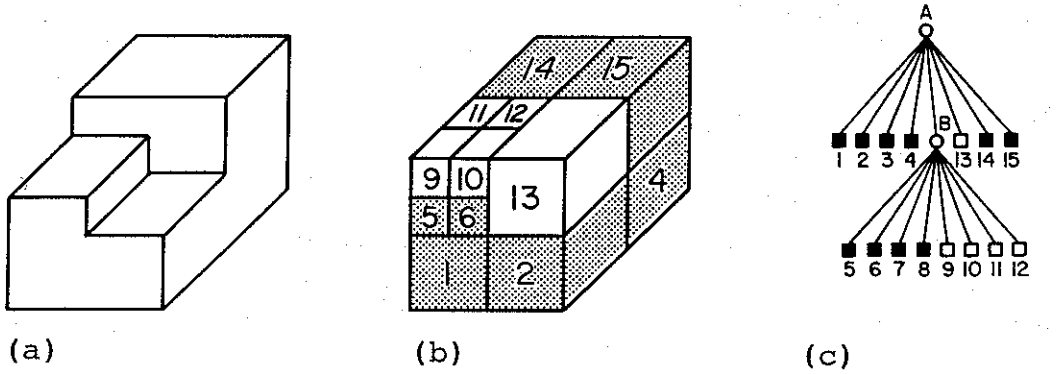

Figure 3. (a) Example three-dimensional object; (b) its octree block decomposition; and (c) its tree representation.

There are a number of different planar decomposition methods. We use a quadtree in the form of squares because it is a planar decomposition that satisfies the following two properties:

(1) It yields a partition that is an infinitely repetitive pattern and thus it can be used for images of any size.

(2) It yields a partition that is infinitely decomposable into increasingly finer patterns (i.e., higher resolution).

A quadtree-like decomposition into four equilateral triangles (see Figure 4a) also satisfies these criteria. However, unlike the decomposition into squares, it does not have a uniform orientation - i.e., all tiles with the same orientation cannot be mapped into each other by translations of the plane which do not involve rotation or reflection. In contrast, a decomposition into hexagons (see Figure 4b) has a uniform orientation but it does not satisfy property (2). For more details on the properties of decompositions see Bell *et al.* [Bell83].

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, termed the *linear quadtree* [Garg82], 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

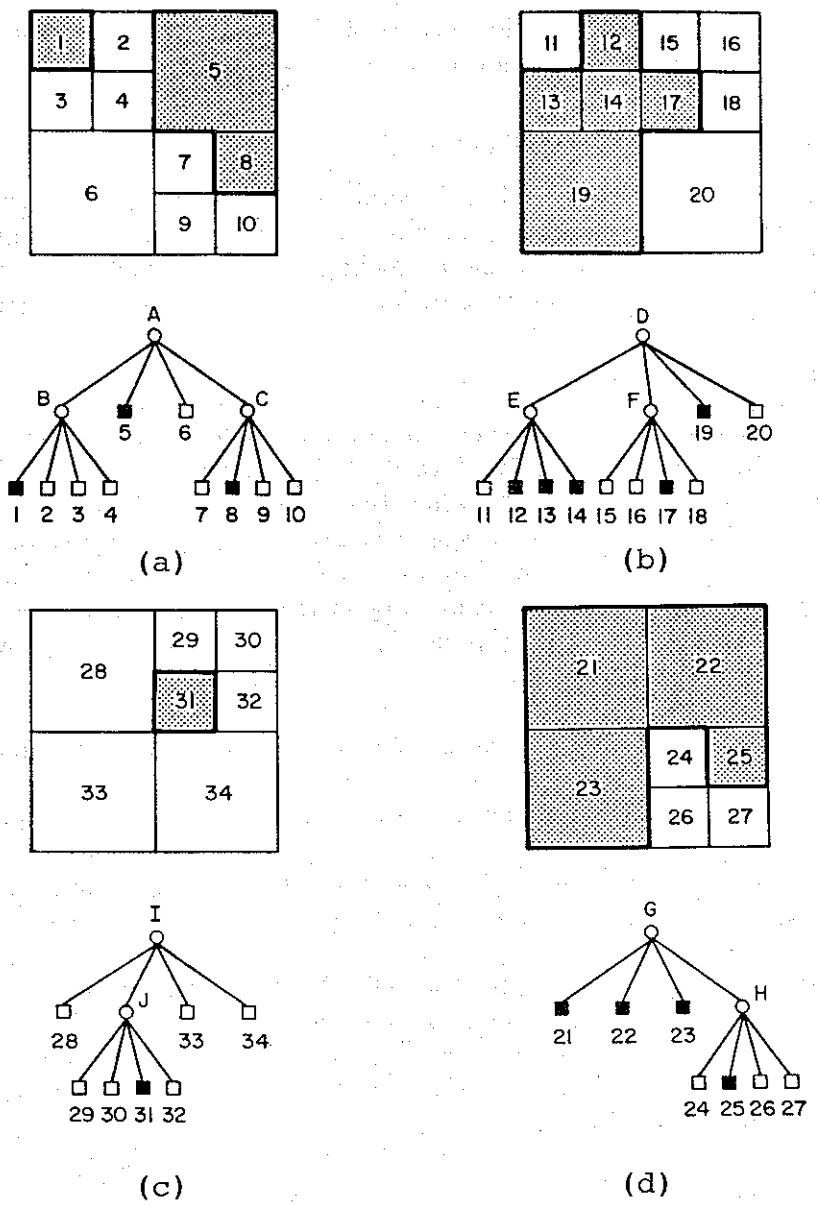

Figure 5. Example of set-theoretic operations. (a) sample image and its quadtree; (b) sample image and its quadtree; (c) intersection of the images in (a) and (b); union of the images in (a) and (b).

The ability to perform set operations quickly is one of the primary reasons for the popularity of quadtrees over alternative representations such as the chain code. The chain code can be characterized as a local data structure, since each segment of the chain code conveys information only about the part of the image to which it is adjacent - i.e., that the image is to its right. Performing an overlay operation on two images represented by chain codes thus requires a considerable amount of work. In contrast, the quadtree is a hierarchical data structure that yields successive refinements at lower levels in the tree.

3.2. BOTTOM-UP NEIGHBOR FINDING

Many quadtree algorithms involve more work than just traversing the tree. In particular, in several applications we must perform a computation at each node that depends on the values of its adjacent neighbors. Thus we must be able to locate these neighbors. There are several techniques for achieving this result. One approach [Klin79] makes use of the coordinates and the size of the node whose neighbor is being sought in order to compute the location of a point in the neighbor. For a $2^n \times 2^n$ image, this can require n steps corresponding to the path from the root of the quadtree to the desired neighbor. An alternative approach, and the one we describe below, only makes use of father links and computes a direct path to the neighbor by following links in the tree. This method is termed *bottom-up neighbor finding* and has been shown to require an average of four links to be followed for each neighbor that is sought [Same82].

In this section we shall limit ourselves to neighbors in the horizontal and vertical direction that are of size equal to or greater than the node whose neighbor is being sought. For neighbors in the diagonal direction, see [Same82]. Finding a node's neighbor in a specified horizontal or vertical direction requires us to follow father links until a common ancestor of the two nodes is found. Once the common ancestor is located, we descend along a path that retraces the previous path with the modification that each step is a reflection of the corresponding prior step about the axis formed by the common boundary between the two nodes. The general flow of such an algorithm is given in Figure 6. For example, when attempting to locate the eastern neighbor of node A (i.e., node G) in Figure 6, node D is the common ancestor of nodes A and G, and the eastern edge of the block corresponding to node A is the common boundary between node A and its neighbor. The main idea behind bottom-up neighbor finding can be understood by examining more closely how the nearest common ancestor of a node, say P, and its eastern neighbor of greater than or equal size, say Q, is located. In particular, the nearest common ancestor has P as one of the eastern-most nodes of one of its western subtrees, and Q as one of the western-most nodes of one of its eastern subtrees. Thus as long as an ancestor X is in a subtree that is not an eastern son (i.e., NE or SE), we must ascend the tree at least one more level before locating the nearest common ancestor.

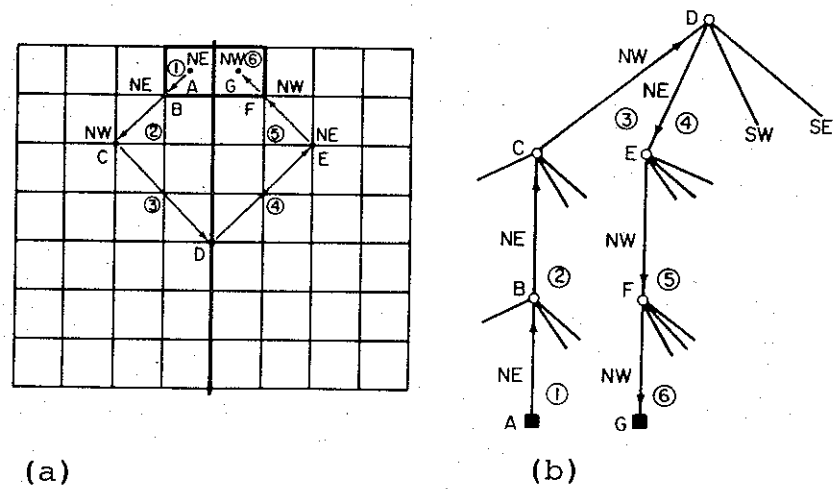

Figure 6. The process of locating the eastern neighbor of node A (i.e., G). (a) Block decomposition; and (b) Tree representation.

southern neighbor, have yet been labeled and thus label B is assigned to them. When block 20 is processed, it has no label but its southern neighbor, 31, has already been assigned B as a label and thus block 20 is assigned label B as well. Second, Figure 8b shows the status of the image at the conclusion of the second step of the algorithm. It has three different labels (i.e., A, B, and C) with B equivalent to C. This equivalence was generated when the eastern adjacency of block 23 was explored. In essence, block 23 was labeled with C when block 22's eastern adjacency was explored, whereas block 31 was labeled with B when block 15's southern adjacency was explored. Thus we see that the third step of the algorithm will have to be applied thereby relabeling all C blocks with B.

The execution time of this connected component labeling algorithm is almost linear in the number of BLACK blocks. Thus it is dependent only on the number of blocks in the image and not on their size. In contrast, the analogous algorithm for the binary array has an execution time that is proportional to the number of pixels and hence to the area of the blocks. Therefore, we see that the hierarchical structure of the quadtree data structure saves not only space but time.

The first step of the connected component labeling algorithm (i.e., the adjacency determination) can be used to compute the perimeter of an image represented by a quadtree [Same81c] as well as component counting [Dyer80]. Component counting is very closely related to the computation of the genus (i.e., Euler number) of an image. In the case of an image represented by a quadtree, the genus is determined by simply computing the quantity $V-E+F$ where V corresponds to the number of BLACK blocks in the image, E is the number of pairs of adjacent BLACK blocks in the horizontal and vertical directions, and F is the number of cases where three or four BLACK blocks touch at a common point. This result is due to Dyer [Dyer80].

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 data and R for region) [Oren82, Same84a]. It is an adaptation of the region quadtree to point data which associates data points 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 9 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 9 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 quadrants of the tree rooted at the SE quadrant (i.e., (75,25)).

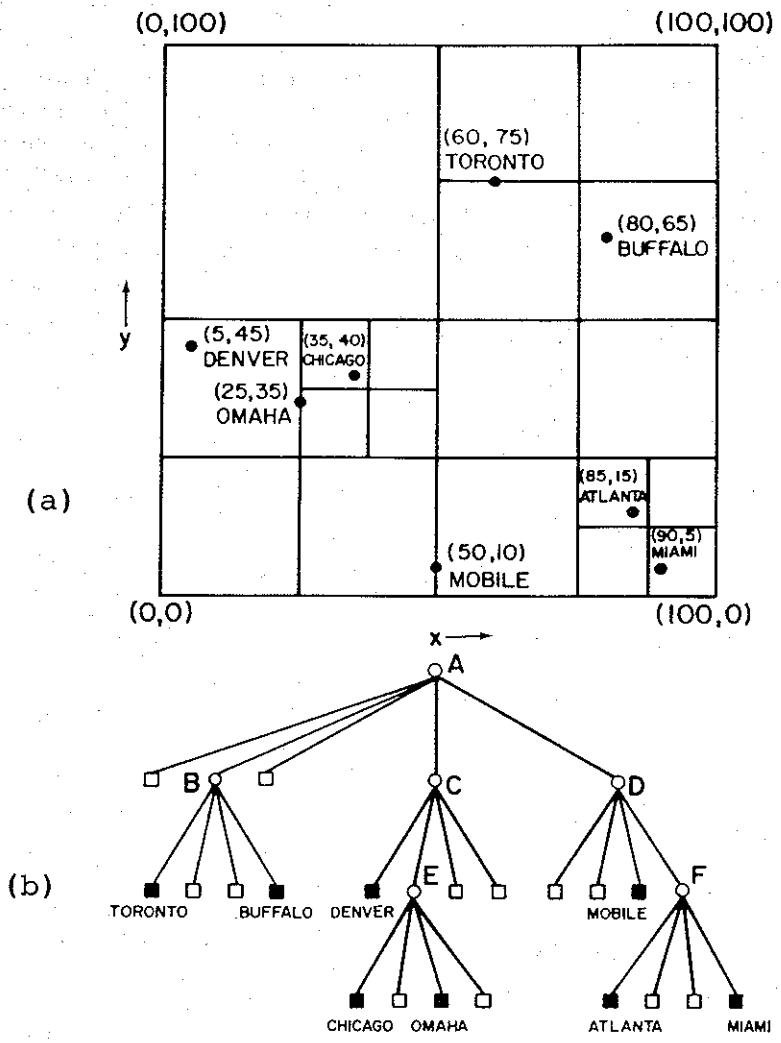

Figure 9. A PR quadtree (b) and the records it represents (a).

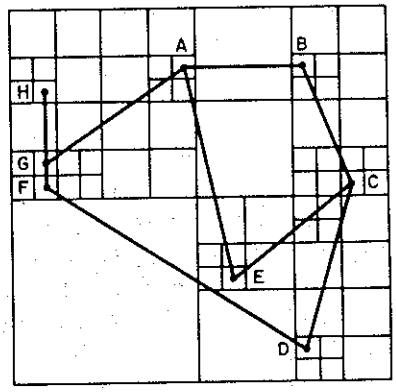

Figure 12. An edge quadtree.

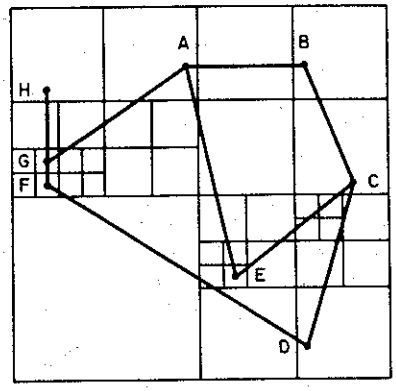

Figure 13. The PM quadtree of Figure 12.

The PM quadtree of Samet and Webber [Same85b] (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 impossible for five or more line segments to meet at a vertex. There are a number of variants of the PM quadtree. The one we describe is based on a decomposition rule that stipulates that partitioning occurs as long as a quadrant contains more than one line segment unless the line segments are all incident at the same vertex which is also in the same quadrant (e.g., Figure 13). We 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). In such a case, 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.

6. CONCLUDING REMARKS

The use of hierarchical data structures in image databases enables us to focus computational resources on the interesting subsets of data. Thus there is no need to expend work where the payoff is small. Moreover, algorithms based on such methods are easy to develop and maintain. When the hierarchical data structures are based on the principle of recursive decomposition, we have the added benefit of a spatial index. All features, be they regions, points, or lines, 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. There are leaf nodes corresponding to region, point, and line data. Point data is implemented using a PR quadtree and line data is implemented using a PM quadtree.

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.

REFERENCES

1. [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).

2. [Bell83] - S.B.M. Bell, B.M. Diaz, F. Holroyd, and M.J. Jackson, Spatially referenced methods of processing raster and vector data, *Image and Vision Computing 1*, 4(November 1983), 211-220.

3. [Dyer80] - C.R. Dyer, Computing the Euler number of an image from its quadtree, *Computer Graphics and Image Processing 13*, 3(July 1980), 270-276.

4. [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.

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

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

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

8. [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.

9. [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.

10. [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.

11. [Kawa83] - E. Kawaguchi, T. Endo, and J. Matsunaga, Depth-first expression viewed from digital picture processing, *IEEE Transactions on Pattern Analysis and Machine Intelligence 5*, 4(July 1983), 373-384.

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

13. [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.

14. [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.

15. [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.

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

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

18. [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).

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

20. [Rose83] - A. Rosenfeld, Ed., *Multiresolution Image Processing and Analysis*, Springer Verlag, Berlin, 1983.

21. [Same80] - H. Samet, Region representation: quadtrees from binary arrays, *Computer Graphics and Image Processing 13*, 1(May 1980), 88-93.

22. [Same81a] - H. Samet, An algorithm for converting rasters to quadtrees, *IEEE Transactions on Pattern Analysis and Machine Intelligence 3*, 1(January 1981), 93-95.

23. [Same81b] - H. Samet, Connected component labeling using quadtrees, *Journal of the ACM 28*, 3(July 1981), 487-501.

24. [Same81c] - H. Samet, Computing perimeters of images represented by quadtrees, *IEEE Transactions on Pattern Analysis and Machine Intelligence 3*, 6(November 1981), 683-687.

25. [Same82] - H. Samet, Neighbor finding techniques for images represented by quadtrees, *Computer Graphics and Image Processing 18*, 1(January 1982), 37-57.

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

27. [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.

28. [Same85b] - 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).

29. [Same86] - H. Samet and R.E. Webber, A comparison of the space requirements of multi-dimensional quadtree-based file structures Computer Science TR-1711, University of Maryland, College Park, MD, September 1986.

30. [Same87] - H. Samet and R.E. Webber, Hierarchical data structures and algorithms for computer graphics, Computer Science TR-1752, University of Maryland, College Park, MD, January 1987.

31. [Shaf87] - C.A. Shaffer and H. Samet, Optimal quadtree construction algorithms, to appear in *Computer Vision, Graphics, and Image Processing.*

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

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

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

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

36. [Warn68] - J.E. Warnock, A hidden line algorithm for halftone picture representation, Computer Science Department TR 4-5, University of Utah, Salt Lake City, May 1968.