On Encoding Boundaries with Quadtrees

HANAN SAMET AND ROBERT E. WEBBER

Abstract—The "line quadtree" data structure hierarchically represents a region and its boundary. Based on the standard quadtree, each node in this structure stores adjacency information as well. While line quadtrees use the same amount of space as standard qaudtrees, they facilitate several region processing algorithms. In particular, we describe efficient algorithms for boundary and superimposing two maps encoded as line quadtrees.

***Index Terms*—Boundary following, boundary representation, hierarchical data structures, image representation, map superimposition, polygonal maps, quadtrees.**

I. Introduction

The quadtree [6], [7] is a data structure for region representation in image processing and geographic information systems that is based on the principle of recursive decomposition. The numerous variants of the quadtree can be differentiated on the basis of the type of data they represent (points, regions, curves, volumes, or surfaces) and on the principle guiding the decomposition process (regular or not). In fact, the term quadtree has taken on a generic meaning to denote a hierarchical data structure adapted to the application of interest.

The two current approaches to region representation specify either the borders or the interior of regions. Common border representations include the chain code [5], the strip tree [1], the BSPR (a bottom-up analog of the strip tree) [2], and quadtree-based methods such as the edge quadtree [11]. Although these representations are suitable for extracting border information, they are difficult to use for performing set operations that involve the interiors of the regions whose boundaries they represent (such as union, superimposing one map on another, and intersection). These operations are important in geographic information systems because they enable the processing of queries like "Find the borders of all wheat growing regions that are located at altitudes between 100 and 200 feet above sea level." On the other hand, the region quadtree is ideal for performing set operations because it hierarchically encodes interiors of regions. To extract boundary information, however, one must use an algorithm, similar to that reported in [3], that traverses a border by visiting each block adjacent to it.

This correspondence describes a new data structure, the line quadtree, that hierarchically specifies both boundary and interior information. Its novelty lies in making use of the nonterminal nodes of the region quadtree to store information about the boundaries spanned by these nodes, which yields efficient boundary following and map superimposition. Section II contains a brief description of the region quadtree. Section III defines the line quadtree and Section IV shows how it can be used to facilitate boundary and map superimposition. For a more detailed explanation of the algorithms for constructing line quadtrees and performing operations such as map superimposition the reader may consult an earlier and longer version of this paper in [9].

Manuscript received July 11, 1983; revised October 13, 1983. This work was supported by the U.S. Army Engineer Topographic Laboratories under Contract DAAK70-81-C-0059.

H. Samet is with the Department of Computer Science and the Center for Automation Research, University of Maryland, College Park, MD 20742.

R. E. Webber was with the Department of Computer Science and the Center for Automation Research, University of Maryland, College Park, MD 20742. He is now with the Department of Computer Science, Rutgers University, New Brunswick, NJ 08903.

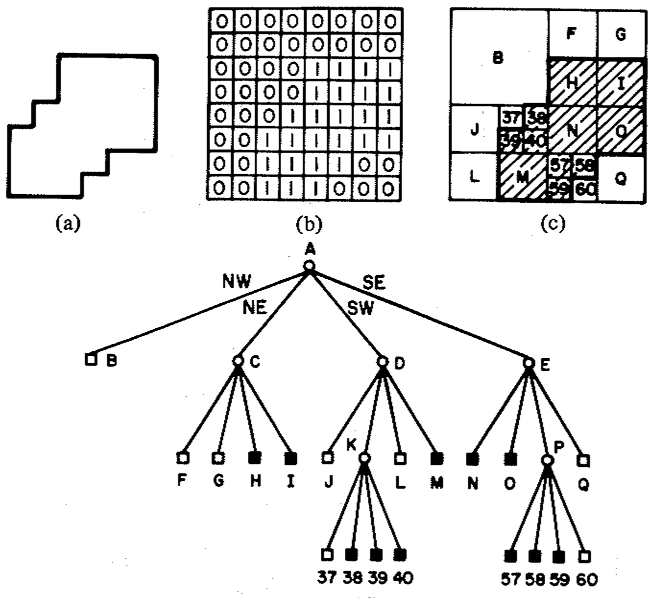


Fig. 1. A region, its binary array, its maximal blocks, and the corresponding quadtree. (a) Region. (b) Binary array. (c) Block decomposition of the region in (a). Blocks in the image are shaded. (d) Quadtree representation of the blocks in (c).

II. Region Quadtrees

The quadtree is an approach to region representation that is based on the successive subdivision of an image array into quadrants. If the array does not consist entirely of 1's (i.e., BLACK pixels corresponding to foreground) or entirely of 0's (i.e., WHITE pixels corresponding to background), then subdivide the array into quadrants, subquadrants, . . . until obtaining square blocks (possible 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. As an example, Fig. 1(b) is an image array corresponding to the region in Fig. 1(a). Fig. 1(c) is the result of applying the block decomposition process to Fig. 1(b). This process can be represented by a tree of out-degree 4. The root node corresponds to the entire array. Each son of a node represents a quadrant (labeled in order NW, NE, SW, SE). The leaf nodes of the tree correspond to those blocks for which no further subdivision is necessary. A leaf node is BLACK or WHITE depending on whether its corresponding block is entirely inside or entirely outside of the represented region. All nonterminal nodes are GRAY. The quadtree representation for Fig. 1(c) is shown in Fig. 1(d).

The region quadtree is different from the point quadtree of [4], which is generalization of the binary search tree [8] to an arbitrary number of dimensions. The point quadtree is designed for storing multidimensional point data and results in a partitioning of the space into regions that are not of equal size.

III. Line Quadtrees

In this paper we address the problem of storing line segments that partition a plane. Since the partitions bounded by the

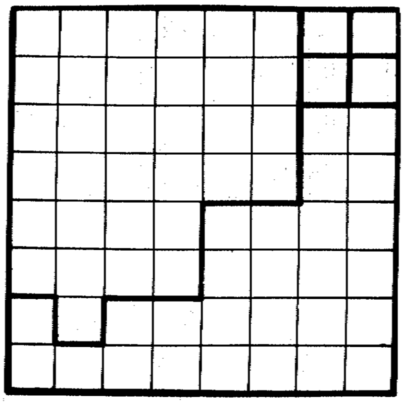


Fig. 2. Picture (map) M on 8 × 8 array of pixels.

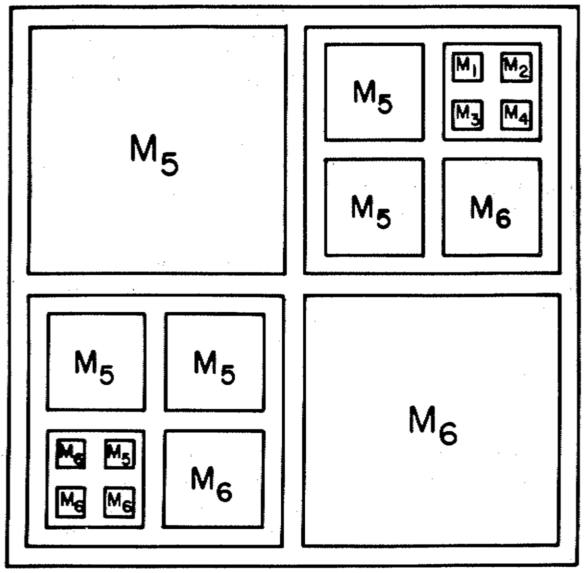


Fig. 3. The region quadtree for M.

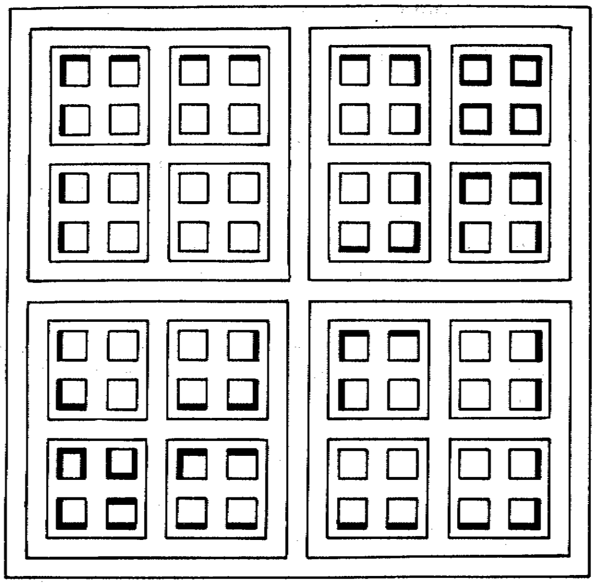


Fig. 4. Complete 4-ary tree for map M.

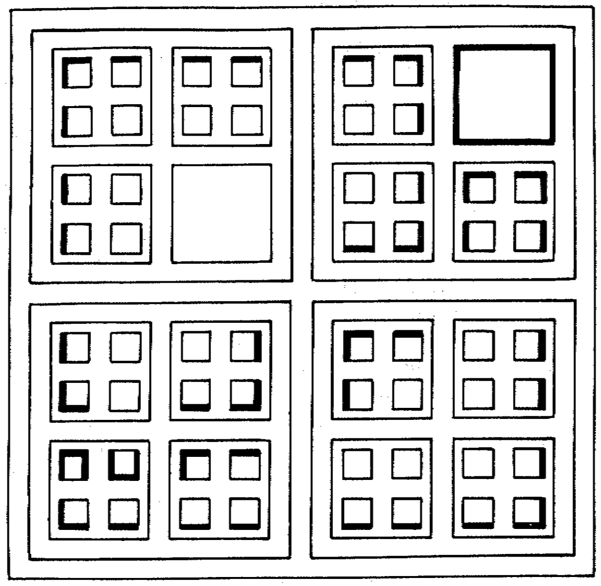


Fig. 5. The quadtree for M produced by the first method.

line segments correspond to regions, we could make use of data structures developed for storing regions. This is analogous to representing a map by assigning each region a separate color. Alternatively, we could represent a map as a two-color region where the boundary lines are represented as narrow BLACK regions and the remainder of the map is colored WHITE. This is awkward for two reasons: 1) narrow regions are costly in terms of the number of nodes in the quadtree and 2) it commits the user to a predefined thickness of the boundary line. The latter may be unfortunate when regenerating the picture on an output device.

Region quadtrees may be defined in a top-down manner (as done in Section II) or in a bottom-up manner. The primitive entity of a bottom-up region quadtree definition is a pixel. Merging is based on the pixel's color. The primitive entity of a bottom-up line quadtree definition is also a pixel. However, merging is based on edge information, i.e., for each pixel the presence or absence of an edge on each of the four sides of the pixel is recorded. Four nodes of identical boundary information are merged to yield a larger node having the same boundary information. As an example, consider the map given in Fig. 2 which decomposes a space into six regions whose region quadtree is given in Fig. 3. The complete 4-ary tree corresponding to Fig. 2 is given in Fig. 4. Edges of pixels that are part of a region partition are marked with a bold line. Applying the merging criterion yields Fig. 5.

As Fig. 5 so vividly demonstrates, our merging criterion is not very attractive. Only two 4-tuples of brothers were found to be suitable for merging. We need a merging criterion that permits four blocks to be merged if the block resulting from their merger has no lines passing through it. Equivalently, in a top-down sense, the map is partitioned via a recursive decomposition technique which successively subdivides it until obtaining blocks (possibly single pixels) that have no line segments passing through their interior. With each leaf node, a code is associated that indicates which of its four sides form a solid border (not a partial border) of any region. This border information is hierarchical in that it is also associated with nonterminal nodes. In essence, whenever a nonterminal node has a side that is fully adjacent to a border, say b, and this

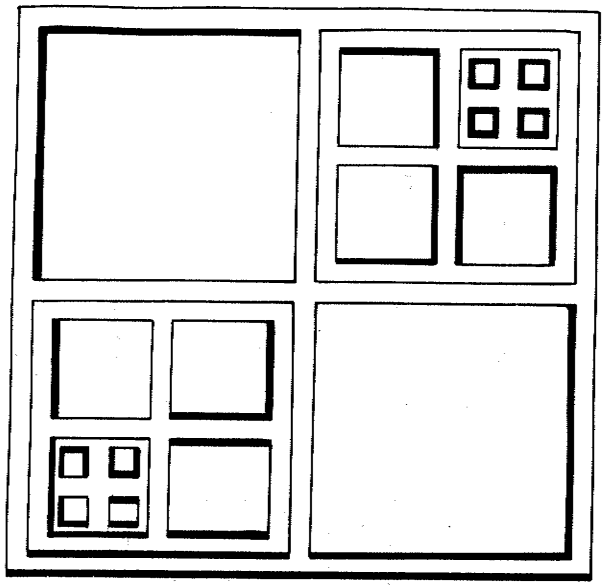


Fig. 6. The line quadtree for *M* produced by the second method.

border does not form a *T*-junction with any of the borders of its descendants along *b*, then this side is marked as a solid border in the nonterminal node.

As an example, consider Fig. 6 which is the line quadtree corresponding to the polygonal map of Fig. 2. The bold lines indicate the presence of borders. Note that the south side of the block corresponding to the root is on a border. On the other hand, the western side of the SW son of the root in Fig. 6 does not indicate the presence of a border even though a border is there. This is due to the existence of a *T*-junction between its NW and SW sons.

The number of nodes in the line quadtree is equal to the number of nodes in the region quadtree. This is obvious when we observe that the top-down line quadtree construction process is equivalent to the top-down region quadtree construction process. Recall that blocks through which line segments pass are subdivided in the latter. This is analogous to a subdivision based on nonhomogeneity of color.

An image can be reconstructed from its line quadtree. This is true even though only full borders (not partial borders) are marked with a bold line (termed SET in contrast to CLEAR which corresponds to a partial border or the absence of a border). For example, the western side of the SE son of the root in Fig. 6 is marked CLEAR. The feasibility of the reconstruction is based on the following two premises: 1) if a terminal node's side is marked SET, then there was an edge of that length in that position in the original image; 2) if a terminal node's side is marked CLEAR, but there are segments of the map that coincide with segments of that side, then those edges will be represented by sides marked SET in the adjacency tree for that side. The adjacency tree [6] of a terminal node x on a given side is a binary tree rooted at the neighbor y (possibly an internal node) on that side and containing all descendants of y that have x as a neighbor. For example, in Fig. 6, we see that a complete description of the W side of the SE son of the root can be obtained by examining the E side of the NE son of the SW son of the root. Since this segment of border does not extend down the entire W side of the SE son of the root, it must turn west (note that all curves are implicitly closed). By turning west, the line prevents the merger of the sons of the SW son of the root, thereby ensuring that one of these unmerged sons could encode it as a side marked SET.

IV. Applications

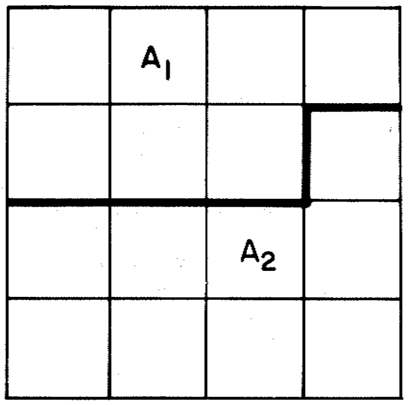
In order to demonstrate the utility of the line quadtree we show how to do boundary following and map superimposition. Boundary following algorithms using the line quadtree are straightforward extensions of the quadtree to boundary code algorithm [3]. The important difference between boundary-following in a line quadtree and boundary-following in a region quadtree, is that in a line quadtree, it is not necessary to straddle the boundary to know that it is there. When following the border of a region, frequently it is only necessary to visit the nodes on one side of the boundary. Hence, the edge-following algorithms can be expected to visit (on the average) half as many nodes when executing on line quadtrees as when executing on region quadtrees. Thus edge following algorthms using a line quadtree should be twice as fast as those that use the region quadtree. Algorithmically, we may view the line quadtree as the result of preprocessing the region quadtree to improve the execution time of boundary following algorithms. As with most preprocessing techniques, its usefulness is determined by the frequency of using the precalculated information.

Actually, boundary following algorithms based on the line quadtree are slightly faster than indicated above. Often, they can move onward on the basis of information stored at an interior node and not have to examine all the terminal nodes along the path. As an example, consider the application of a boundary following algorithm to the bottom edge of the map of Fig. 2 with the aid of the line quadtree of Fig. 6. Such an algorithm knows that the S side of the root is solid and thereby never has to descend to the SE son of the SW son of the SW son of the root to verify the presence of that segment of the edge.

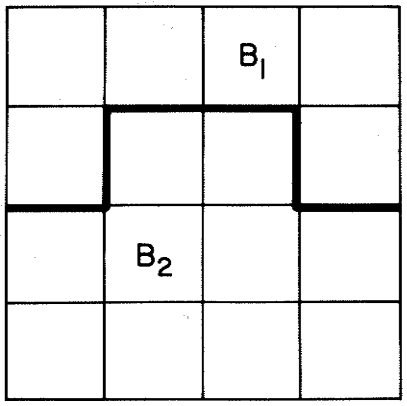
As mentioned in Section I, map superimposition (i.e., overlaying two images) is an important task in a cartographic information system. As an example, consider the two maps in Fig. 7: map *A* is composed of two regions ($A1$ and $A2$) and map *B* is composed of two regions ($B1$ and $B2$). Their corresponding line quadtrees are shown in Fig. 8. The result of the map superimposition operation is shown in Fig. 9. Note that if two cells are in separate regions in either map, then they will also be in separate regions in the result of map superimposition.

Map superimposition using line quadtrees is a three-step process. First, construct a new tree, say *C*, by performing a logical OR of corresponding elements of the two line quadtrees. This is achieved by a postorder traversal of the two trees, say *A* and *B*, in parallel and adding nodes corresponding to nonterminal nodes in *C*. When a leaf node in one of the two trees, say node *D* in *A*, is encountered, the remaining subtree of *B*, say *E*, is copied to *C*. The edge information of the copy of *E* is set to the logical OR of the edges common to both *E* and *D*. Note that internal nodes do not have their edge information updated at this point. For example, consider Fig. 10 which shows the result of the application of this step to the line quadtrees of Fig. 8. The second step corrects erroneous edge information for terminal nodes. These are edges which were marked CLEAR in the original line quadtrees but should be marked SET as a result of map superimposition. As an example, consider the northern edge of the SE son of the root in Fig. 10. The result of this step is shown in Fig. 11. The third and final step is to propagate the edge information upward in the tree to the internal nodes. Fig. 12 shows the line quadtree resulting from superimposition of the line quadtrees of Fig. 8.

The execution time of the map superimposition algorithm is equal to the sum of the execution time of the three steps. The first and third steps are tree traversals and their execution time is proportional to the number of nodes in the trees. The second step requires examination of all adjacent nodes on the four sides of each leaf node. This can be achieved in time pro-



(a)



(b)

Fig. 7. (a) Map A. (b) Map B.

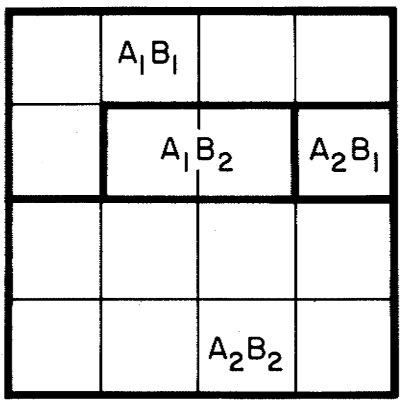


Fig. 9. Map AB.

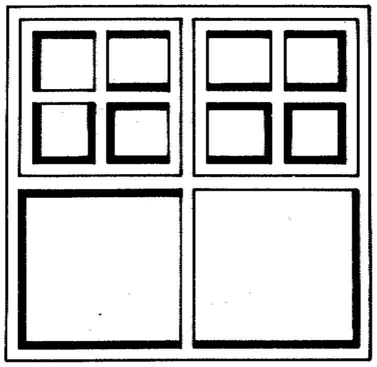
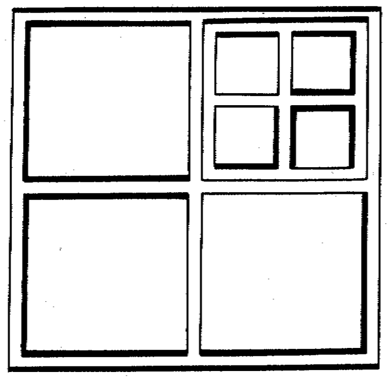
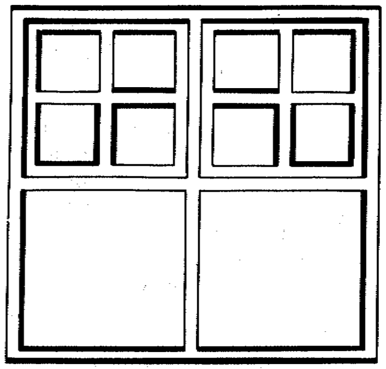


Fig. 10. Result of OR'ing together the two quadtrees of maps A and B.



(a)



(b)

Fig. 8. (a) Corresponding line quadtree for Fig. 7(a). (b) Corresponding line quadtree for Fig. 7(b).

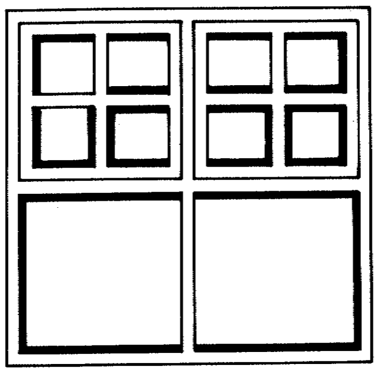


Fig. 11. Result of correcting leaves in tree of Fig. 10.

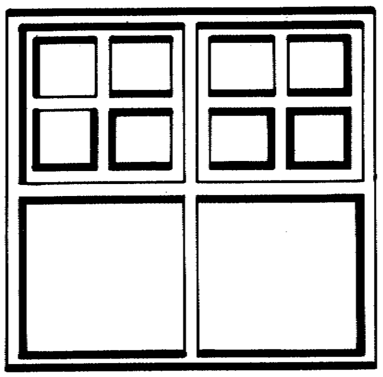


Fig. 12. Result of merging leaves in tree of Fig. 11.

portional to the number of nodes in the tree by passing the neighbors as parameters to the tree traversal algorithm. For details see [10]. Thus we see that map superimposition for line quadtrees can be achieved in time proportional to the number of nodes in the trees.

V. Concluding Remarks

We have presented a hierarchical data structure for storing maps and their boundaries. The novelty of the approach lies in using the internal nodes to store information about the boundaries. The utility of the data structure has been demonstrated by showing how common operations such as boundary following and map superimposition are efficiently performed.

References

[1] D. H. Ballard, "Strip trees: A hierarchical representation for curves," *Commun. Ass. Comput. Mach.*, vol. 24, pp. 310-321, May 1981; see also "Corrigendum," *Commun. Ass. Comput. Mach.*, vol. 25, p. 213, Mar. 1982.

[2] W. Burton, "Representation of many-sided polygons and polygonal lines for rapid processing," *Commun. Ass. Comput. Mach.*, vol. 20, pp. 166-171, Mar. 1977.

[3] C. R. Dyer, A. Rosenfeld, and H. Samet, "Region representation: Boundary codes from quadtrees," *Commun. Ass. Comput. Mach.*, vol. 23, pp. 171-178, Mar. 1980.

[4] R. A. Finkel and J. L. Bentley, "Quad trees: A data structure for retrieval on composite keys," *Acta Informatica*, vol. 4, pp. 1-9, 1974.

[5] H. Freeman, "Computer processing of line-drawing images," *ACM Comput. Surveys*, vol. 6, pp. 57-97, Mar. 1974.

[6] G. M. Hunter and K. Steiglitz, "Operations on images using quadtrees," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-1, pp. 145-153, Apr. 1979.

[7] A. Klinger, "Patterns and search statistics," in *Optimizing Methods in Statistics*, J. S. Rustagi, Ed. New York: Academic, 1971, pp. 303-337.

[8] D. E. Knuth, *The Art of Computer Programming—Vol. 1: Fundamental Algorithms*, 2nd ed. Reading, MA: Addison-Wesley, 1975.

[9] H. Samet and R. E. Webber, "Line quadtrees: A hierarchical data structure for encoding boundaries," Dep. Comput. Sci., Univ. Maryland, College Park, TR-1162, Feb. 1982.

[10] H. Samet, "A top-down quadtree traversal algorithm," Dep. Comput. Sci., Univ. Maryland, College Park, TR-1237, Dec. 1982.

[11] M. Shneier, "Two hierarchical linear feature representations: Edge pyramids and edge quadtrees," *Comput. Graphics Image Processing*, vol. 17, pp. 211-224, Nov. 1981.