Correspondence

An Algorithm for Converting Rasters to Quadtrees

HANAN SAMET

Abstract—**An algorithm is presented for constructing a quadtree for a binary image given its row-by-row description. The algorithm processes the image one row at a time and merges identically colored sons as soon as possible, so that a minimal size quadtree exists after processing each pixel. This method is spacewise superior to one which reads in an entire array and then attempts to build the quadtree.**

Index Terms—**Image processing, pattern recognition, quadtrees, region representation.**

I. Introduction

Region representation is an important issue in image processing, cartography, and computer graphics, and consequently a number of representations are currently in use (see [1] for a brief review). In this correspondence we present an algorithm for obtaining an in-core quadtree representation [2], [3] given the row-by-row description of a binary image. The algorithm is useful because each of the two representations is well-suited for a particular set of operations or may be desirable for its compactness. For example, the row-by-row representation is especially useful for interaction with raster-like display devices, since input and output requires very little additional computation. In addition, it is also a useful technique when memory size limitations preclude storing in core an array corresponding to the image (e.g., [10]). On the other hand, the quadtree is a compact hierarchical representation, thereby facilitating search. For related work dealing with reorganization of raster-scan image data for auxiliary storage see [4].

Assume that the image is a $2^n \times 2^n$ array. Each row of the image is thus a bit string of length 2^n. The quadtree is an approach to image representation based on successive subdivision of the image into quadrants. In essence, we repeatedly subdivide the array into quadrants, subquadrants, . . . , until we obtain blocks (possibly single pixels) which consist entirely of either 1's or 0's. This process is represented by a tree of out-degree 4 in which the root node represents the entire array, the four sons of the root node represent, in order, the NW, NE, SW, and SE quadrants, and the terminal nodes correspond to those blocks of the array for which no further subdivision is necessary. For example, Fig. 1(b) is a block decomposition of the region in Fig. 1(a), while Fig. 1(c) is the corresponding quadtree. In general, BLACK and WHITE square nodes represent blocks consisting entirely of 1's and 0's, respectively. Circular nodes, also termed GRAY nodes, denote nonterminal nodes.

Manuscript received June 13, 1979; revised December 19, 1979. This work was supported by the Defense Advanced Research Projects Agency and the U.S. Army Night Vision Laboratory under Contract DAAG-53-76C-0138 (DARPA Order 3206).

The author is with the Department of Computer Science, University of Maryland, College Park, MD 20742.

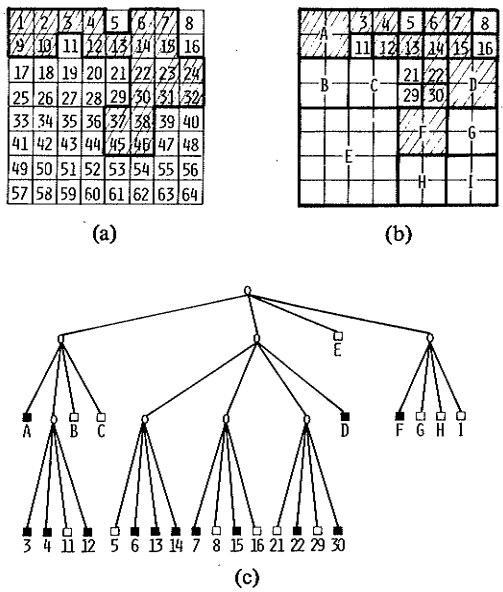

Fig. 1. An image, its maximal blocks, and the corresponding quadtree. Blocks in the image are shaded. (a) Sample image. (b) Block decomposition of the image in (a). (c) Quadtree representation of the blocks in (b).

II. Algorithm

The raster-to-quadtree algorithm processes the image by rows starting with the first row. One of its key features is that at any instant of time (i.e., after each pixel in a given row has been processed) a valid quadtree exists with all unprocessed pixels presumed to be WHITE. Thus, as the quadtree is built, nodes are merged to yield maximal blocks. This is in contrast with an algorithm that first builds a complete quadtree with one node per pixel and then attempts to merge, i.e., replace all GRAY nodes with four sons of the same color by a node of the same color. The disadvantage of the complete quadtree method is that it requires more space. In particular, for a $2^n \times 2^n$ image, 2^{2n} BLACK and WHITE nodes may be required in addition to $1/3 \cdot 2^{2n}$ nonterminal GRAY nodes. This is clearly undesirable when compared with a maximum of 2^{2n} bits required by the binary array representation and the fact that a significant amount of merging is expected to take place. Note that a hybrid method was used in [6] to construct a quadtree from the boundary code of the image, i.e., the first pass left a number of links unspecified, while the second pass filled in the links and attempted to merge the resulting nodes.

Recall that the given binary image is assumed to be partitioned into rows. Clearly, no odd-numbered row can lead to a merge of nodes. Thus, odd-numbered rows do not require as much processing as even-numbered rows. We further assume that the image contains an even number of rows. If the image contains an odd number of rows, then it is presumed that one extra row of WHITE has been added.

For an odd-numbered row, the tree is constructed by pro-

0162-8828/81/0100-0093$00.75 © 1981 IEEE

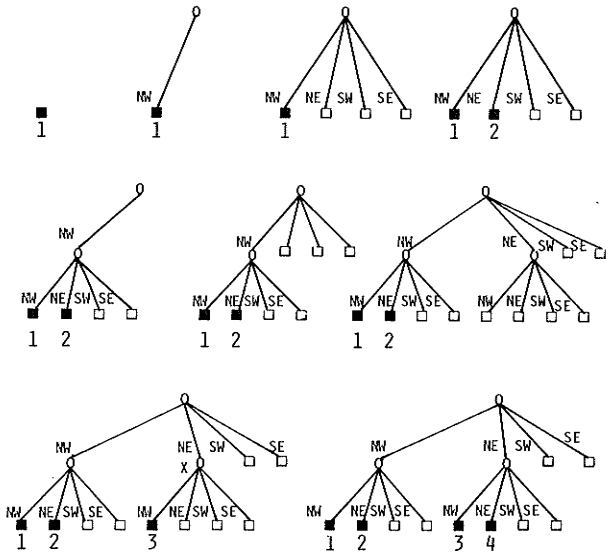

Fig. 2. Intermediate trees in the process of obtaining a quadtree for the first part of the first row in Fig. 1(a).

cessing the row from left to right adding a node to the tree for each pixel. For example, Fig. 2(a)–(i) shows the construction of a quadtree corresponding to the first four pixels of the binary image of Fig. 1(a) (i.e., pixels 1, 2, 3, and 4). This is done by invoking a procedure described below, called ADD–NEIGHBOR. As the quadtree is constructed, nonterminal nodes must also be added. Since we wish to have a valid quadtree after processing each pixel, whenever we add a nonterminal node we also add, as is appropriate, three or four WHITE nodes as its remaining sons.

We now describe ADD–NEIGHBOR more formally. Adding a neighbor of a node, say P, in a specified direction consists of traversing ancestor links until a common ancestor of the two nodes is found. Once the common ancestor is found, we descend along a path that retraces the previous path with the modification that each step is a reflection of the corresponding step about the axis formed by the common boundary between the two nodes. For example, when attempting to add the eastern neighbor of node 3 (i.e., node 4) in Fig. 2(h), node X is the common ancestor and the eastern edge of the block corresponding to node 3 is the common boundary. Thus, having ascended a NW link to reach node X, reflection about the eastern edge of node 3's block causes us to descend to the NE son of X. If a common ancestor does not exist, then a nonterminal node is added with its three remaining sons being WHITE [e.g., Fig. 2(c) and (f)]. Once the common ancestor and its three sons have been added, we once again descend along the retraced path modified by reflecting each step about the axis formed by the boundary of the node whose neighbor we seek. During this descent, a WHITE node is converted to a GRAY node and four WHITE sons are added [e.g., Fig. 2(g)]. As a final step, the terminal node is colored appropriately [e.g., Fig. 2(d) and (h)]. In the example, Fig. 2(a), (b)–(d), (e)–(h), and (i) are snapshots of the quadtree construction process for the nodes corresponding to pixels 1, 2, 3, and 4, respectively, of Fig. 1(a).

Even-numbered rows require more work since merging may also take place. In particular, a check for a possible merge must be performed at every even-numbered vertical position (i.e., every even-numbered pixel in a row). Once a merge occurs, we may have to check if another merge is possible. In particular, for pixel position $(a \cdot 2^i, b \cdot 2^j)$ where $a \bmod 2 = b \bmod 2 = 1$, a maximum of $k = \min(i, j)$ merges is possible. For example, at pixel 60 of Fig. 1(a), i.e., position (8, 4), a maximum of 2 merges is possible and indeed this is how block

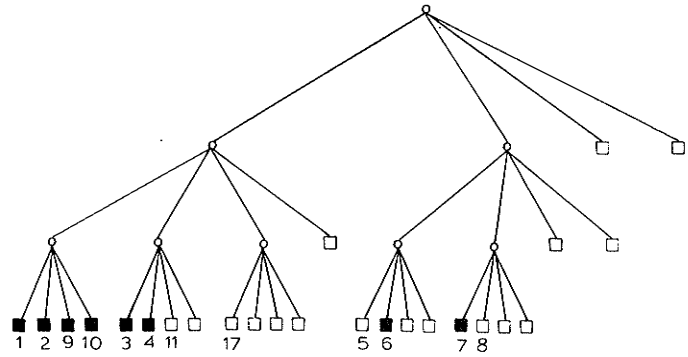

Fig. 3. Quadtree prior to merging nodes 1, 2, 9, and 10.

E of Fig. 1(b) has been obtained. The fact that merging does take place causes an additional amount of bookkeeping. In particular, we wish to maintain the position in the tree where the next pixel is to be added as well as the next row. Therefore, prior to attempting a merge, a node corresponding to the next pixel in the image is added to the quadtree [e.g., node 11 is added to the quadtree in Fig. 3 prior to attempting to merge nodes 1, 2, 9, and 10 of Fig. 1(a)]. Similarly, we precede the processing of each even-numbered row by adding to the quadtree a node corresponding to the first pixel in the next row [e.g., the addition of node 17 to the tree of Fig. 3 prior to processing row 2 of Fig. 1(a)]. This type of look-ahead was also employed for different reasons in algorithms for computing the perimeter of a region [7] and labeling connected components [8].

For a more detailed presentation of the algorithm see [9], where Algol-like procedures are given to perform the conversion. As an example of the algorithm, consider the image given in Fig. 1(a). Fig. 1(b) is the corresponding block decomposition and Fig. 1(c) is its quadtree representation. The nodes labeled A–I are a result of merging and the alphabetical order corresponds to the order in which the merged nodes were created. Fig. 2(a)–(i) shows the steps in the construction of the quadtree corresponding to the first part of the first row. Figs. 4 and 5 show the resulting tree after the first and second rows have been processed.

III. Concluding Remarks

An algorithm has been presented for converting a row-by-row representation of a binary image into a quadtree representation of the image. In [9] it is shown that the algorithm's execution time has a time complexity proportional to the number of pixels in the image. This is obtained by examining the number of nodes that are visited as the tree is constructed. In particular, the number of nodes visited by the merging process is bounded by the number of pixels in the image while the remaining part of the tree construction process visits four times as many nodes as there are pixels in the image (see [9]). Note that the algorithm is also spacewise efficient in that merging is attempted whenever possible. Thus, after processing each pixel in a given row the resulting quadtree contains a minimal number of nodes.

The algorithm is one-dimensional in the sense that it processes the image a row at a time. Thus, it can be used in conjunction with the run length representation [5], which views each row as a sequence of maximal runs of pixels having the same value. Therefore, a row is completely determined by specifying the lengths of these runs and the value of the first run. When only a few runs are present, this representation is very economical. For example, consider the image in Fig. 1(a). Its run length coding is B4121, B2141, W53, W53, W422, W8, W8, where commas serve as separators between rows.

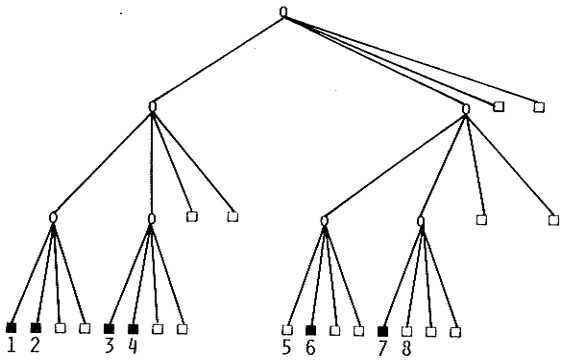

Fig. 4. Quadtree after processing the first row in Fig. 1(a).

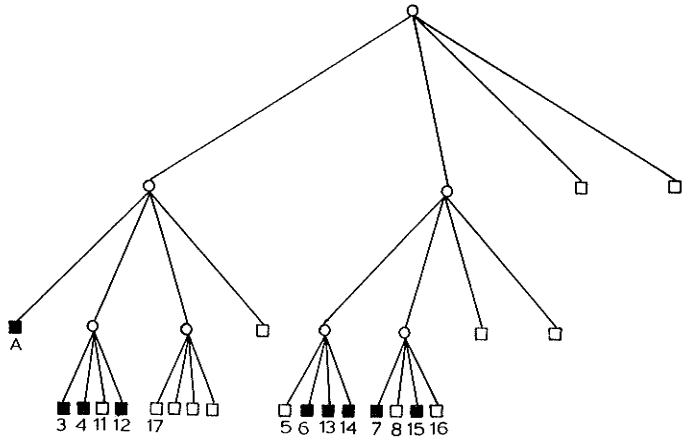

Fig. 5. Quadtree after processing the second row in Fig. 1(a).

The algorithm can be contrasted with two other approaches. If sufficient memory is available to store the array in core, then the technique of [10], which only creates nodes corresponding to maximal blocks, and hence requires no merging, should be used. Alternatively, we could attempt to find maximal blocks by processing several rows at once (e.g., 2^m consecutive rows with runs of length $\geq 2^m$ at the NW-most corner of the image yield a block of size 2^m). The disadvantage of such an approach is that it requires searching and a substantial amount of bookkeeping as the rows are being processed. Instead, we have found maximal blocks by merging at appropriate pixel positions.

Acknowledgment

The author would like to thank K. Riley for her help in preparing the manuscript and P. Young for drawing the figures. The author has also benefited from discussions with C. R. Dyer and A. Rosenfeld.

References

[1] C. R. Dyer, A. Rosenfeld, and H. Samet, "Region representation: Boundary codes from quadtrees," *Commun. Ass. Comput. Mach.*, vol. 23, pp. 171–179, Mar. 1980.

[2] 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.

[3] A. Klinger and C. R. Dyer, "Experiments in picture representation using regular decomposition," *Comput. Graphics and Image Process.*, vol. 5, pp. 68–105, 1976.

[4] A. Klinger and M. L. Rhodes, "Organization and access of image data by areas," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-1, pp. 50–60, Jan. 1979.

[5] A. Rosenfeld and A. C. Kak, *Digital Picture Processing*. New York: Academic, 1976.

[6] H. Samet, "Region representation: Quadtrees from boundary codes," *Commun. Ass. Comput. Mach.*, vol. 23, pp. 163–170, Mar. 1980.

[7] —, "Computing perimeters of images represented by quadtrees," Dep. Comput. Sci., Univ. of Maryland, College Park, MD, Tech. Rep. 755, Apr. 1979.

[8] —, "Connected component labeling using quadtrees," Dep. Comput. Sci., Univ. of Maryland, College Park, MD, Tech. Rep. 756, Apr. 1979; also in *J. Ass. Comput. Mach.*, to be published.

[9] —, "Region representation: Raster-to-quadtree conversion," Dep. Comput. Sci., University of Maryland, College Park, MD, Tech. Rep. 766, May 1979.

[10] —, "Region representation: Quadtrees from binary arrays," *Comput. Graphics Image Process.*, vol. 13, pp. 88–93, 1980.