A Top-Down Quadtree Traversal Algorithm

HANAN SAMET

Abstract—Many standard image processing operations can be implemented using quadtrees as a simple tree traversal where, at each terminal node, a computation is performed involving some of that node's neighbors. Most of this work has involved the use of bottom-up neighbor-finding techniques which search for a nearest common ancestor. Recently, top-down techniques have been proposed which make use of a neighbor vector as the tree is traversed. A simplified version of the top-down method for a quadtree in the context of a general-purpose tree traversal algorithm is presented. It differs, in part, from prior work in its ability to compute diagonally adjacent neighbors rather than just horizontally and vertically adjacent neighbors. It builds a neighbor vector for each node using a minimal amount of information. Analysis of the algorithm shows that its execution time is directly proportional to the number of nodes in the tree. However, it does require some extra storage. Use of the algorithm leads to lower execution time bounds for some common quadtree image processing operations such as connected component labeling.

Index Terms—Connected component labeling, image processing, image representation, perimeter, quadtrees.

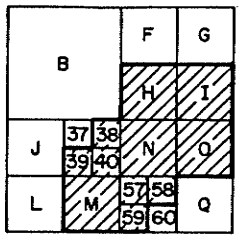
I. Introduction

The quadtree [7] (e.g., Fig. 1) is a hierarchical representation which has been the subject of much research in recent years [15]. It has been found to be useful in such applications as image processing, computer graphics, pattern recognition, and cartography. Many algorithms for standard operations in these domains can be expressed as simple tree traversals where at each node a computation is performed involving the use of "bottom-up" neighbor-finding techniques [11]. Recently, "top-down" methods which build a neighbor vector as the tree is traversed have been independently proposed [5], [8], [13].

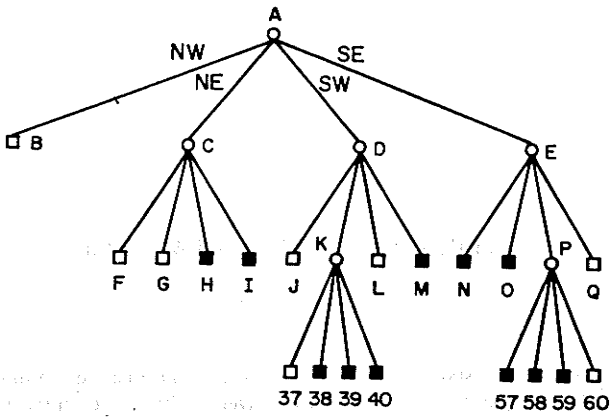
Manuscript received December 7, 1983; revised July 23, 1984. Recommended for acceptance by S. L. Tanimoto. This work was supported by the U.S. Army Engineer Topographic Laboratories under Contract DAAG-70-81-C-0059.

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

0162-8828/85/0100-0094$01.00



(a) A region and its block decomposition. Blocks in the region are shaded.



(b) Quadtree representation of the region in (b).

Fig. 1. A region and its corresponding quadtree.

In this paper, we present a simplified formulation of the top-down method for a quadtree in the context of a general-purpose tree traversal algorithm. It differs, in part, from prior work in its ability to compute diagonally adjacent neighbors rather than just horizontally and vertically adjacent neighbors. The execution time and storage cost associated with the top-down technique are analyzed. We also show how it can be used to speed up a number of common quadtree operations. In particular, for N BLACK blocks, connected component labeling is shown to be $O(N \cdot \log N)$, whereas using bottom-up techniques, it has a worst case of at least $O(N^2)$. The conclusion contains a comparison of our method to that of [5].

II. Algorithm

A natural byproduct of the tree-like nature of the quadtree representation is that many basic operations are implemented as tree traversals. The difference between them is in the nature of the computation that is performed at the node. Often, these computations involve the examination of some nodes that are adjacent to the node being processed. We shall speak of these adjacent nodes as neighbors. In order to be more precise, given node P, corresponding to block P, and a direction D, we say that node Q, corresponding to block Q, is the *neighbor* of node P in direction D (i.e., *neighbor* $(P, D) = Q$) when both of the following conditions are satisfied.

1) P and Q share a common border, even if only a corner.

2) The block corresponding to Q is the smallest block (it may be GRAY) of size greater than or equal to the block corresponding to P.

For example, block N in Fig. 1 has neighbors H, I, O, Q, P, M, K, and B in the directions N, NE, E, SE, S, SW, W, and NW, respectively. Note that the neighboring nodes need not be distinct, i.e., a node may serve as a neighbor in more than one direction. For example, for node 37 in Fig. 1, node B overlaps the NW, N, and NE neighboring directions; node J overlaps the W and SW directions; while the remaining neighbors are nodes 38, 40, and 39 in the E, SE, and S directions, respectively.

As mentioned above, most operations are implemented as tree traversals with the operation being performed by examining the neighbors of selected nodes in the tree. In order for the operation to be performed in the most general manner, we must be able to locate neighbors in a way that is independent of both position (i.e., the coordinates) and the size of the node. We also do not want to use any additional links to adjacent nodes. In other words, only the structure of the tree should be used, and no pointers in excess of the four links from a node to its four sons and one link to its father for a nonroot node. This is in contrast to the methods of Klinger and Rhodes [7] which make use of size and position information, and those of Hunter and Steiglitz [4] which locate neighbors through the use of explicit links (termed *ropes* and *nets*).

In [11], neighbors are located by ascending the tree until a common ancestor is located, and then descending down the tree in search of the neighboring node. This technique, termed *bottom-up*, has an average worst case of four nodes being visited for each neighbor that is sought. The worst case for a $2^n \times 2^n$ image is $2 \cdot n$. An alternative method, termed *top-down* and presented below, is one that transmits a neighbor vector, as an actual parameter, containing the eight neighbors of each node in the eight directions as the tree is traversed. A one-dimensional variant of this technique for horizontal and vertical neighbors is used by Jackins and Tanimoto [5] in the computation of perimeter for K-dimensional objects. It has also been used in [8] in an empirical comparative study of neighbor-finding techniques, and in [13] for superposing polygonal maps. Such techniques are termed "top-down" in contrast to the "bottom-up" method which works by locating the nearest common ancestor.

Procedure TRAVERSAL, given below using a variant of Algol 60, incorporates the top-down method in enabling the application of an arbitrary function F to every terminal node in the quadtree. We assume a quadtree implementation using pointers, i.e., a nonterminal node contains four pointers to its four sons and accessed by the field SON. For each son, say Q, of a nonterminal node, say P, TRAVERSAL computes a neighbor vector consisting of the eight neighbors of Q. It is interesting to observe that given a direction or side D, we compute all of the neighbors by the appropriate application of the functions OPSIDE, CSIDE, CCSIDE, and QUAD. In particular, OPSIDE(D) yields the side facing D, CSIDE(D) and CCSIDE(D) correspond to the sides adjacent to D in the clockwise and counterclockwise directions, respectively, and QUAD($S1, S2$) is the quadrant bounded by sides $S1$ and $S2$ of a block. Note the use of the function SONI instead of SON when computing the elements of the neighbor vector that do not correspond to brothers (e.g., the northern neighbor of node K in Fig. 1 when attempting to compute the neighbor vector for node 38, the NE son of K).

As an example of the execution of TRAVERSAL, consider the quadtree of Fig. 1. Initially, TRAVERSAL must be invoked with a neighbor vector consisting of all NIL entries since we are dealing with a root node, i.e., A. Recursively, applying TRAVERSAL to node A leads us first to the computation of the neighbor vector of node C (i.e., node A's NE son) which consists of NIL, NIL, NIL, NIL, E, D, B, and NIL in the directions N, NE, E, SE, S, SW, W, and NW, respectively. Since C is a nonterminal node, we recursively apply TRAVERSAL to the four sons of C. For example, the resulting neighbor vector for H, the SW son of C, is F, G, I, O, N, K, B, and B in the directions N, NE, E, SE, S, SW, W, and NW, respectively. At this point, we can apply function F to node H and its appropriate neighbors.

```
procedure TRAVERSAL (P, L, A, F);
/* Given a quadtree rooted at node P spanning a 2^L X 2^L
   space, apply function F to each of its terminal nodes. A
```

```
    contains the adjacent neighbors of P of greater than or
    equal size in the directions N, NE, E, SE, S, SW, W, and
    NW. Initially, A contains 8 NIL pointers for the neigh-
    bors of the root of the quadtree. GRAY(P) is false when
    P is NIL.*/
begin
  value node P;
  value integer L;
  reference node array A;
  function F;
  direction D;
  node array T;
  if GRAY(P) then /* Descend a level in the tree*/
    begin
      for D in {"N," "E," "S," "W"} do
        begin
          T[D] ← SONI(A[D], QUAD(OPSIDE(D),
            CSIDE(D)));
          T[QUAD(D, CSIDE(D))] ← SONI(A[QUAD
            (D, CSIDE(D))],
            QUAD(OPSIDE(D), CCSIDE(D)));
          T[CSIDE(D)] ← SONI(A[CSIDE(D)], QUAD
            (D, CCSIDE(D)));
          T[QUAD(OPSIDE(D), CSIDE(D))] ← SONI(A
            [CSIDE(D)],
            QUAD(OPSIDE(D), CCSIDE(D)));
          T[OPSIDE(D)] ← SON(P, QUAD(OPSIDE(D),
            CSIDE(D)));
          T[QUAD(OPSIDE(D), CCSIDE(D))] ← SON(P,
            QUAD(OPSIDE(D), CCSIDE(D)));
          T[CCSIDE(D)] ← SON(P, QUAD(D, CCSIDE
            (D)));
          T[QUAD(D, CCSIDE(D))] ← SONI(A[D],
            QUAD(OPSIDE(D), CCSIDE(D)));
          TRAVERSAL(SON(P, QUAD(D, CSIDE(D))),
            L-1, T, F);
        end;
      end
    else APPLY(F, P, L, A); /* Apply F to terminal node
      P*/
  end;

node procedure SONI(P, Q);
/* Return a pointer to the son of node P in quadrant Q. If
   node P is a terminal node, then return P.*/
begin
  value node P;
  value quadrant Q;
  return (if GRAY(P) then SON(P, Q)
          else P);
end;
```

III. Analysis

The execution time of the traversal algorithm can be obtained by counting the number of nodes that are visited or accessed during the tree traversal and neighbor vector computation respectively. First, let us prove the following lemma.

Lemma 1: For a quadtree with N nodes of which G are GRAY, $N = 4 \cdot G + 1$.

Proof: Each GRAY node has an out degree of 4. The quadtree has $4 \cdot G$ edges. It is well known that a tree with $4 \cdot G$ edges has $4 \cdot G + 1$ nodes. Clearly, $N = 4 \cdot G + 1$.

We now come to the main result.

Theorem 1: The number of nodes visited or accessed by procedure TRAVERSAL is bounded by $9 \cdot N$.

Proof: For each GRAY node, procedure TRAVERSAL constructs four neighbor vectors, each of which involves the access of eight nodes. This yields a total of $8 \cdot 4 \cdot G$ nodes. But from Lemma 1, $N = 4 \cdot G + 1$. Therefore, $8 \cdot (N - 1)$ nodes are accessed. Procedure TRAVERSAL also visits each of the N nodes in the tree in the process of constructing the neighbor vectors. Therefore, the total number of nodes that are visited or accessed is $8 \cdot (N - 1) + N = 9 \cdot N - 8$. •

The maximum amount of storage required by procedure TRAVERSAL is obtained as follows.

Theorem 2: For a $2^n \times 2^n$ image represented by a quadtree, procedure TRAVERSAL requires at most $8 \cdot (n + 1)$ storage units.

Proof: Starting at the root of the quadtree, assumed to be at level n, it takes i recursive calls of TRAVERSAL to reach a node at level n-i. Each recursive invocation of TRAVERSAL requires eight storage units for the neighbor vector. Since we also need an initial neighbor vector, a node at level n-i needs $8 \cdot (i + 1)$ storage units. A $2^n \times 2^n$ image may have a terminal node at level 0 (i.e., a pixel) which requires n recursive invocations of TRAVERSAL. Thus, we need $8 \cdot (n + 1)$ storage units. •

IV. Applications

The top-down neighbor computation method, in conjunction with the traversal algorithm, can be used to implement a number of quadtree operations. Jackins and Tanimoto [5] used a variant of it to compute the perimeter for K-dimensional objects. Their method works by performing K tree traversals (i.e., one for each dimension). Depending on the way the perimeter problem is posed, this involves two or four neighbors. Using our technique, the perimeter can be computed by simply using the southern and eastern components of the neighbor vector and searching for BLACK-WHITE and WHITE-BLACK adjacencies [10]. This approach was used in [8]. Of course, once the neighbor has been located, if it corresponds to a GRAY node, then all nodes adjacent to the common border must be visited (i.e., the adjacency tree). This process is bounded since the total number of nodes in all the adjacency trees for a given quadtree is bounded by four times the number of leaf nodes in the quadtree [3]. Connected component labeling [9] can be performed the same way, except that now we are searching for BLACK-BLACK adjacencies. On the other hand, all eight components of the neighbor vector are necessary for the computation of the Chessboard distance transform for quadtrees [12], the construction of a quadtree medial axis transform (QMAT) from a quadtree [14], and the reverse process of reconstructing a quadtree from its QMAT [16].

Use of the top-down neighbor computation method in conjunction with the traversal algorithm of Section III leads to lower execution time bounds for a number of quadtree operations which involve neighbor finding. In these cases, the cost of neighbor finding is part of the overall cost of the operation, and speeding it up may result in achieving a lower execution time. The bottom-up neighbor-finding process is analyzed in [11] where for each BLACK node, the average number of nodes visited while seeking a neighbor in a given direction is shown to be constant. However, the worst case for an image containing N BLACK blocks may be $O(N^2)$ as shown below.

Theorem 3: The worst case for bottom-up neighbor finding for an image containing N BLACK blocks is at least $O(N^2)$.

Proof: Consider a $2^N \times 2^N$ image of the form of Fig. 2 ($N = 4$ in this case) that contains N BLACK blocks.[1] Computing the eastern neighbor of the BLACK blocks requires

$$\sum_{i=1}^{N} i + 1 = \frac{1}{2} \cdot N \cdot (N + 3)$$

nodes to be visited. •

Knowing that the worst case of bottom-up neighbor finding is at least $O(N^2)$, while top-down neighbor finding is $O(N)$,

[1] Note that this example is different from that of [5] which exhibits $N \cdot \log N$ behavior in an $N \times N$ image with N BLACK blocks.

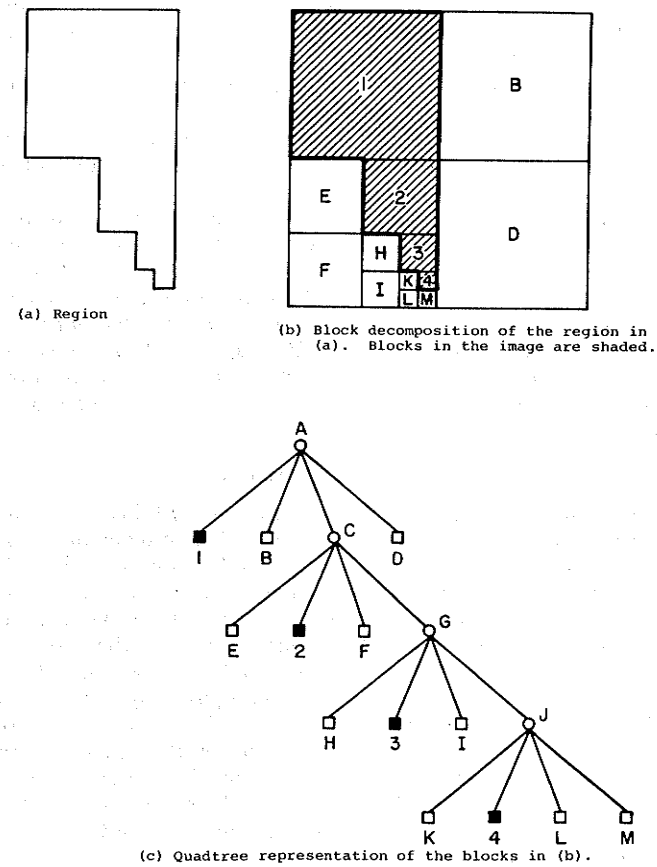


Fig. 2. A quadtree corresponding to an image which leads to the worst case behavior of the quadtree-connected component labeling algorithm's first phase.

enables us to obtain a lower bound for the connected component labeling algorithm for images represented by quadtrees. Recall that in [9], it was shown that the average execution time for connected component labeling is $O(N+N \cdot \log N)$. The algorithm was shown to have three steps. The first step involved examination of the southern and eastern neighbors of each BLACK node, i.e., an $O(N)$ process on the average which, at times, can be $O(N^2)$. The second step involved propagation of equivalence classes which can be done in $O(N \cdot \log N)$ time. In fact, almost linear behavior can be obtained by combining the first and second steps using the *UNION-FIND* algorithm [17]. The third step requires updating equivalences, which is an $O(N)$ process. Thus, we see that the bottleneck is the first step which, by use of the top-down method, is $O(N)$. Therefore, the worst case for connected component labeling is now almost $O(N)$ instead of at least $O(N^2)$.

A similar analysis can be applied to the perimeter computation algorithm of [10], and also to the Euler number computation method of [2]. Both of these algorithms perform tree traversals where at each BLACK terminal node, a number of adjacent neighbors are examined. In essence, they are analogous to the first step of the connected component labeling of [9]. Using the top-down algorithm of Section III results in an $O(N)$ algorithm instead of an average $O(N)$ algorithm. Note that Jackins and Tanimoto [5] obtain the same time bound for computing the perimeter, although their algorithm works by performing K tree traversals (i.e., one for each dimension or two for a quadtree). However, their technique only works for horizontally or vertically adjacent neighbors, and thus cannot be used for corner or diagonally adjacent neighbors.

The order of the execution time of the Chessboard distance transform algorithm of [12] cannot be lowered by use of the top-down method since its computation time is not dominated by the cost of neighbor finding. Similarly, the order of the execution time of the process of reconstructing a quadtree from its QMAT [16] cannot be lowered for the same reason. Nevertheless, the use of the top-down method may lead to an actual reduction in the absolute execution time since the neighbor-finding component is still faster. However, once the distance transform of a quadtree is known, the QMAT can be computed in $O(N)$ time since its computation only requires that for each BLACK block, all eight adjacent neighbors be examined.

V. Concluding Remarks

Our method is different from that of Jackins and Tanimoto [5] in that it is a generalization for all possible neighbors in a quadtree. They show how to compute the perimeter of a K-dimensional object through the use of top-down neighbor-finding techniques. They present a general solution for K dimensions which requires K passes over the data where each pass transmits two neighbors as parameters. Using their technique, only horizontally or vertically adjacent neighbors can be obtained, whereas our method requires only one pass, and most importantly, it also yields the diagonal neighbors. However, our technique is only presented for two-dimensional data. Our method differs from [8] and [13] in the generality of the computation of the neighbor vector for a node of arbitrary type (i.e., it can be a NW, NE, SW, or SE son), and again, in the ability to handle diagonally adjacent neighbors.

As was seen in Sections III and IV, the use of top-down neighbor-finding methods results in the execution time of the neighbor-seeking phase of a number of quadtree algorithms being proportional to the number of nodes in the tree instead of just so on the average. Nevertheless, the bottom-up methods may still be superior at times for a number of reasons. First, less overhead is associated with them since the neighbors may be very close (in a node distance sense). Second, there is no need to compute neighbors of GRAY nodes as is the case for the top-down method. Third, some quadtree operations are inherently bottom-up, such as obtaining a boundary code from a quadtree [1]. Recall that this procedure outputs the chain code as it wanders along the boundary of each region. Bottom-up methods are superior for such a task. Finally, less storage is required by the bottom-up method since there is no need for the neighbor vectors at each level of the tree.

Acknowledgment

I have benefitted greatly from discussions with R. E. Webber. In particular, I would like to thank him for suggesting the possibility of improving the execution time of the connected component labeling algorithm.

References

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

[2] C. R. Dyer, "Computing the Euler number of an image from its quadtree," *Comput. Graphics Image Processing*, vol. 13, pp. 270–276, July 1980.

[3] G. M. Hunter, "Efficient computation and data structures for graphics," Ph.D. dissertation, Dep. Elec. Eng. Comput. Sci., Princeton Univ., Princeton, NJ, 1978.

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

[5] C. Jackins and S. L. Tanimoto, "Quad-trees, oct-trees, and k-trees—A generalized approach to recursive decomposition of Euclidean space," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-5, pp. 533–539, Sept. 1983.

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

[7] 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.
[8] A. Rosenfeld, H. Samet, C. Shaffer, and R. E. Webber, "Application of hierarchical data structures to geographical information systems," Univ. Maryland, College Park, Comput. Sci. TR-1197, June 1982.
[9] H. Samet, "Connected component labeling using quadtrees," *J. ACM*, vol. 28, pp. 487–501, July 1981.
[10] ——, "Computing perimeters of images represented by quadtrees," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-3, pp. 683–687, Nov. 1981.
[11] ——, "Neighbor finding techniques for images represented by quadtrees," *Comput. Graphics Image Processing*, vol. 18, pp. 37–57, Jan. 1982.
[12] ——, "Distance transform for images represented by quadtrees," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-4, pp. 298–303, May 1982.
[13] H. Samet and R. E. Webber, "On encoding boundaries with quadtrees," Univ. Maryland, College Park, Comput. Sci. TR-1162, Feb. 1982.
[14] H. Samet, "A quadtree medial axis transform," *Commun. ACM*, vol. 26, pp. 680–693, Sept. 1983.
[15] ——, "The quadtree and related hierarchical data structures," *ACM Comput. Surveys*, vol. 16, June 1984.
[16] ——, "Reconstruction of quadtrees from quadtree medial axis transforms," *Comput. Vision, Graphics, Image Processing*, to be published; also Univ. Maryland, College Park, Comput. Sci. TR-1224.
[17] R. E. Tarjan, "Efficiency of a good but not linear set union algorithm," *J. ACM*, vol. 22, pp. 215–225, Apr. 1975.

0162-8828/85/0100-0098$01.00