Space and Time Efficient Virtual Quadtress

LESLIE P. JONES AND S. SITHARAMA IYENGAR

Abstract—The quadtree has recently become a major data structure in image processing. This correspondence investigates ways in which quadtrees may be efficiently stored as a forest of quadtrees and as a new structure we call a compact quadtree. These new structures are called virtual quadtrees because the basic operations we expect to perform in moving about within a quadtree can also be performed on the new representations. Space and time efficiency are investigated and it is shown these new structures often given an improvement in both.

Index Terms—Algorithm, data structure, forest, image processing, quadtree.

I. Introduction

Quadtrees have recently been promoted as a method of representation of planar regions. The basic relationship between a region and its quadtree representation is presented in Hunter and Steiglitz [3]. They also discuss several ways of manipulating quadtree representations of regions. Pavlidis [12] describes a method of approximation of pictures by using quadtrees. A good history of the development of quadtrees for representing of a region is available in papers of Klinger, Dyer, and Alexandridis [1], [5]. Samet [7] has given an algorithm for creating a quadtree from an edge-code (chain code), a geometric representation of a region. Conversely, an algorithm to convert an edge-code to a quadtree has been developed by Dyer, Rosenfeld, and Samet [2]. For additional information on manipulating quadtree representations of regions refer to Samet's papers [8]–[10]. For overviews of related research on image processing data structures see [11], [13].

This correspondence investigates ways in which quadtrees may be more efficiently stored as other data structures. Parts of this correspondence review and refine [4]. In Section II we present a structure called a compact quadtree. This structure contains all the information contained in a quadtree but requires much less space. Section III presents a means of converting a quadtree to a forest of quadtrees, also with space savings. We also consider the time efficiency of algorithms that operate on our representations.

For the purposes of this paper, a region will be the BLACK portion of a $2^n \times 2^n$ array made up of a unit square pixels colored BLACK or WHITE. A sample region is presented in Fig. 1 and its quadtree is given in Fig. 2. We will define a node in a quadtree to be a record containing six fields. If P is a (pointer to a) node and D is in the set of directions {NW, NE, SW, SE}, then we may define the fields as follows:

COLOR(P)	has value WHITE or BLACK for a leaf, GRAY for an interior node.
SON(P, D)	(pointer to) the son of P in direction D; NIL if no such node exists.
FATHER(P)	(pointer to) the father of P; NIL if P is the root.

Manuscript received November 15, 1982; revised May 17, 1983.

L. P. Jones was with the Department of Computer Science, Louisiana State University, Baton Rouge, LA 70803. He is now with the Department of Computer Science, Marietta College, Marietta, OH 45750.

S. S. Iyengar is with the Department of Computer Science, Louisiana State University, Baton Rouge, LA 70803.

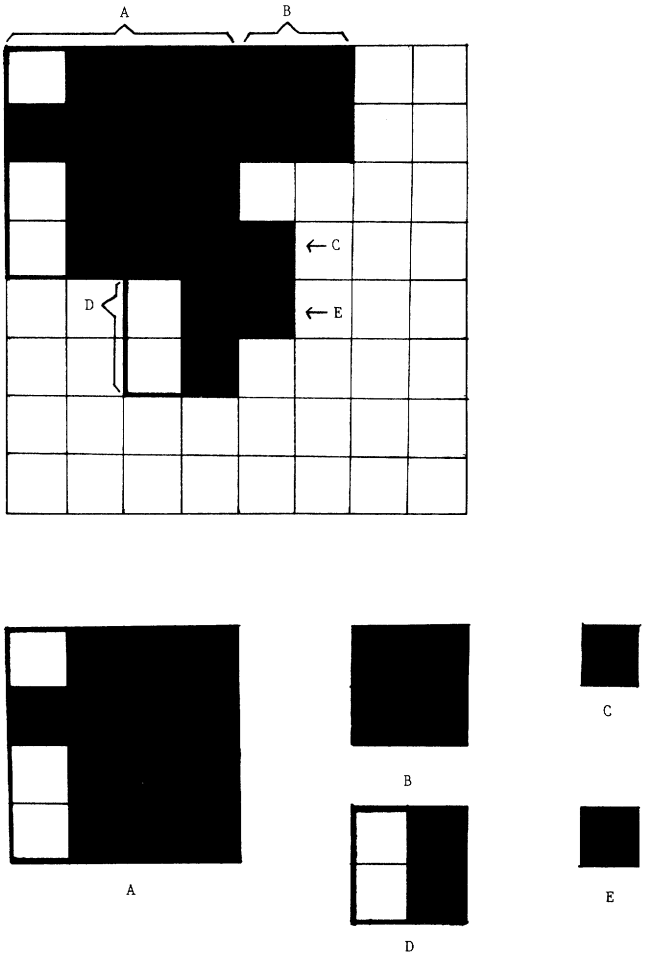

Fig. 1. Region with subregions.

The (pointer to the) root of the quadtree will be denoted by ROOT. We say that the offspring of a node refine that node. In our figures, the offspring of a node are drawn in the canonical order NW–NE–SW–SE.

We also have a need to refer to nodes by position. For this purpose we note that any node in a quadtree may be given unique two-dimensional coordinates (L, K) in the following way. We give the directions, NW, NE, SW, and SE numerical values 0, 1, 2, and 3, respectively. The root has coordinates (1, 0). If P has coordinates (L, K), then its son in direction $D \in \{NW, NE, SW, SE\}$ has coordinates $(L + 1, 4K + D)$. Clearly, L is just the level of P. As an example, Fig. 2 also includes the coordinates of some selected nodes. This coordination scheme is analogous to that of Pavlidis [11, p. 105]. Given the coordinates, (L, K), of a node, the sequence of directional choices on the path from the root to that node may be recovered in the following way. First, we decompose K (uniquely) as a sum of the form

$$K = \sum_{i=0}^{L-2} n_i 4^i \qquad 0 \leqslant n_i \leqslant 3.$$

Next we convert the sequence of integers $(n_{L-2}, n_{L-3}, \cdots, n_0)$ back into a sequence of directions. For example, given the node W in Fig. 2 with coordinates (4, 39), we write

$$39 = (3)4^0 + (1)4^1 + (2)4^2$$

to get the sequence (2, 1, 3) or (SW, NE, SE).

0162-8828/84/0300-0244$01.00

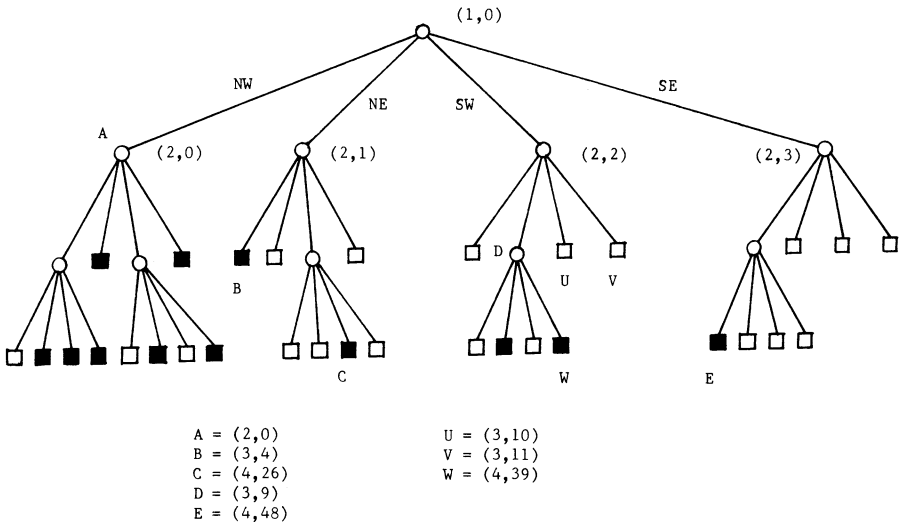

Fig. 2. Quadtree with coordinates.

Let (L, K) and (L′, K′) be (the coordinates of) two nodes in T with $L \leqslant L'$. We note that (L′, K′) is a descendant of (L, K) if and only if

$$K4^{(L'-L)} \leqslant K' < K4^{(L'-L)} + 4^{(L'-L)} = (K+1)4^{L'-L},$$

For example, the leftmost descendant of (2, 3) at level 4 has second coordinate $(3)4^2 = 48$, while the rightmost descendant has second coordinate $(3+1)4^2 - 1 = 64 - 1 = 63$. Clearly, (L′, K′) lies to the left of (L, K), and is not a descendant of (L, K), if and only if

$$K' < K4^{(L'-L)}$$

while (L′, K′) lies the right of (L, K), and is not a descendant of (L, K), if and only if

$$K' \geqslant (K+1)4^{L'-L}.$$

Compare Fig. 2.

As noted, our space efficient ways of representing a quadtree are examples of structures we call virtual quadtrees. A virtual quadtree is any structure which simulates a quadtree in the sense that we can

1) determine the color of any node in the quadtree;
2) find the offspring in any direction of any node in the quadtree;
3) find the father of any node in the quadtree.

II. Compact Quadtrees

First we present a structure called a compact quadtree. If T is a quadtree then we will denote the compact quadtree associated with T by C(T). Each set of four brothers in T is represented by a single record, called a metanode, in C(T). We say that the metanode refines the father of the four nodes. The fields of a metanode M are as follows:

MCOLOR(M, D)	for $D \in \{NW, NE, SW, SE\}$, the colors of the nodes that M represents;
MSONS(M)	(pointer to) the first metanode that represents offspring of a node in T which is represented in M; NIL if no such metanode exists;
MFATHER(M)	(pointer to) the metanode that holds the representation of the father of the nodes that M represents, M's mfather;
MCHAIN(M)	described below.

If there are several metanodes that represent offspring of nodes represented by a given metanode, then these metanodes

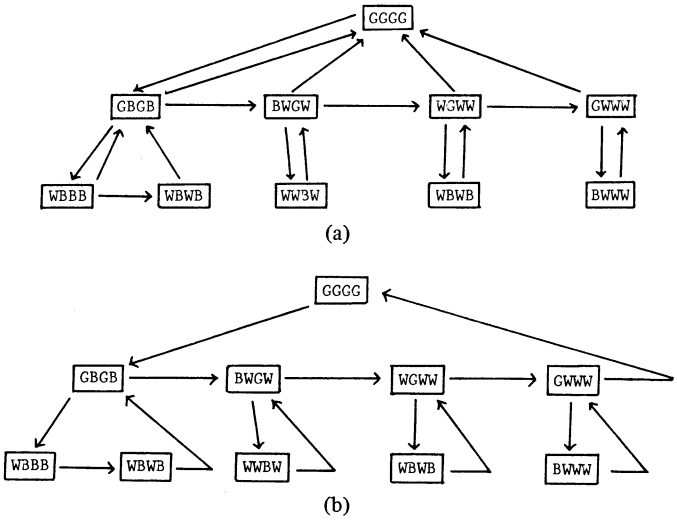

Fig. 3. (a) Compact quadtree C(T). (b) Compact quadtree C(T), without MFATHER links.

are linked via the field MCHAIN. Their order in this linked list is the same as the order of the GRAY fields in M that they refine. The compact quadtree for the quadtree in Fig. 2 is given in Fig. 3(a). Downward links are MSON links, horizontal links are MCHAIN links and upward links are MFATHER links. Note that the GRAY MCOLOR fields of a metanode M are in a natural one-to-one correspondence with metanodes in the chain pointed at by MSONS(M).

The compact quadtree uses the same amount of space as the quadtree for storage of colors but it uses far less space for storage of pointers. In our example, the quadtree of Fig. 2 has 41 nodes or 205 pointers. The compact quadtree has 10 metanodes or 30 pointers, a space savings of about 85 percent. This savings can be expected for any quadtree because its corresponding compact quadtree will have about 1/4 as many records and each record will have 3/5 as many pointers.

If we are willing to accept a small degradation in the time efficiency of finding the mfather of a metanode M, then we can delete the MFATHER field from all the metanodes and merely direct the MCHAIN field of the last metanode in a chain of siblings to indicate the mfather of all the metanodes in that chain. Fig. 3(b) shows the compact quadtree of Fig. 3(a) altered in this manner. This representation requires that each metanode M have a 2-bit field MTOEND(M). If there are

i nodes in the chain after node M, then MTOEND(M) contains i. In this way, we can tell if MCHAIN(M) points to its next sibling or to M's mfather, and we can identify the GRAY field in M's mfather that M refines.

Note that if we represent a quadtree as a binary tree [6] then the space savings is only about 40 percent and the operation of finding of offspring of a node may also require searching as many as four nodes. The much greater space savings comes from the fact that we have also removed a level from the tree.

Naturally, we expect the chaining in the compact quadtree to cause a performance degradation. Curiously enough, a simple recursive tree traversal in order NW-NE-SW-SE may be performed more efficiently on a compact quadtree because there are fewer metanodes and hence fewer subroutine invocations and links followed. The space savings will also produce a time savings on minicomputers with small address spaces and on virtual memory systems because the amount of pictorial data that can be kept in core is far greater for a compact quadtree, thus reducing the number of disk accesses.

III. Forest of Quadtrees

We next consider a virtual quadtree which can provide a large space saving when a region nearly fits into a $2^{n-1} \times 2^{n-1}$ square or when a picture has several widely separated BLACK regions. Let T be a quadtree. A forest, F(T), of quadtrees that represents T consists of a table of triples of the form (P, L, K) and a collection of quadtrees where:

1) each triple (P, L, K) (in the table) consists of the coordinates, (L, K), of a node in T, and a pointer, P, to a quadtree (in the collection) which is identical to the subtree rooted at position (L, K) in T;

2) if (L, K) and (M, N) are coordinates of nodes recorded in F(T), then neither node is a descendant of the other;

3) every BLACK leaf in T is represented by a (BLACK) leaf in F(T).

For example, Fig. 4 contains a forest that represents the tree of Fig. 2. Note that the forest of Fig. 4 corresponds to a decomposition of the region of Fig. 1 into the subregions shown in Fig. 1. We will give an algorithm to reduce a tree T to a forest of quadtrees that represents the tree and we will reconstruct the original tree T from the forest.

We temporarily add a field TYPE(P) to each tree node P. (We assume that space for this field is kept separately from the space for P itself.) The TYPE of P is defined recursively as follows:

1) the TYPE of a BLACK leaf is GOOD;

2) if an interior node has 2 or more GOOD offspring, then it is GOOD;

3) all other nodes are BAD.

The forest that represents T consists of all its maximal subtrees with GOOD roots, i.e., all the subtrees with the property that their root is GOOD but all ancestors of their root in T are BAD.

Let ROOT be (a pointer to) the root of quadtree. The conversion to a forest of quadtrees is accomplished by the sequence of subroutine calls LABEL(ROOT) and FOREST(ROOT, 1, 0), where the subroutines are described below. The first subroutine traverses the entire tree depth-first and determines the TYPE of each node. The second subroutine performs a depth-first traversal locating maximal GOOD nodes and placing the subtrees they root into the forest. BAD nodes encountered by FOREST are freed. The algorithms may be applied to the quadtree of Fig. 2 to get the forest of Fig. 4.

The subroutines are given below in Pascal (we use the notation given previously for references to fields). We also use the symbolic constants ROOT, GOOD, BAD, BLACK, WHITE, and GRAY, all with their obvious meaning. The subroutine FOREST calls the procedure CREATE(P, L, K) which places the triple (P, L, K) in the forest and sets FATHER(P) to NIL. The call DISPOSE(P) frees the node P. Thus the nodes in F(T) are a subset of the nodes of T. If, for any reason, this is undesirable, the algorithm may be easily modified to copy subtrees of T and place the copies in the forest.

Table

2	0	pointer to A
3	4	pointer to B
4	26	pointer to C
3	9	pointer to D
4	48	pointer to E

Trees in forest

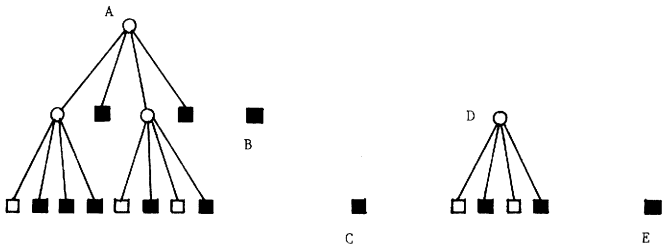

Fig. 4. Forest of quadtrees F(T).

Algorithm—Creation of a Forest of Quadtrees:

```
FUNCTION LABEL(P:NODE): INTEGER;
  (*DETERMINES TYPE(P)*)
  VAR
    D, VALUE: INTEGER;
  BEGIN
    CASE COLOR(P) OF
      BLACK: TYPE(P) := GOOD;
      WHITE: TYPE(P) := BAD:
      GRAY:
        BEGIN
          VALUE := 0;
          FOR D := 0 TO 3 DO
            IF LABEL (SON(P, D))=GOOD THEN
              VALUE := VALUE+1;
          IF VALUE >=2 THEN TYPE(P) := GOOD
                       ELSE TYPE(P) := BAD;
        END (*GRAY*)
    END; (*CASE*)
LABEL := TYPE(P)
END; (*LABEL*)

PROCEDURE FOREST (P: NODE; L, K: INTEGER);
  VAR
    D: INTEGER;
  BEGIN
    IF TYPE(P)=GOOD THEN CREATE(P, L, K);
    ELSE
      BEGIN
        FOR D := 0 TO 3 DO
          FOREST(SON(P, D), L+1, 4*K+D);
        DISPOSE(P)
      END;
  END; (*FOREST*)
```

The time required for the execution of the conversion is obviously linear in the number of nodes in the quadtree. The following can be shown by a simple inductive proof and establishes a maximum size for the table of F(T).

Theorem: The maximum number of trees in a forest derived from a quadtree that represents a square of dimension $2^k \times 2^k$ is 4^{k-1}, i.e., one-fourth the area of the square.

Given a forest of quadtrees F, we can easily show how to reconstruct a quadtree. The reconstructed quadtree R(F) consists of real nodes (nodes in the forest) and virtual nodes (nodes that correspond to BAD nodes deleted while creating the forest). Since virtual nodes require no storage they are located by giving their coordinates. We denote the virtual node with coordinates (L, K) by v(L, K).

The root of R(F) is either a real node (if the forest has one tree) or the virtual node v(1, 0). In either case we know its location and color in R(F).

The offspring of any real node are found by following links. If v(L, K) is any virtual node, then its children in directions $D \in \{NW, NE, SW, SE\}$ have coordinates (L + 1, 4K + D), respectively. It is a simple matter of a table lookup to see if the offspring are real nodes or virtual nodes. Also, we can easily determine the color of a virtual node. It is GRAY if it has a descendant(s) in the table and WHITE otherwise. The check to see if the node has a descendant in the table may be performed efficiently if the table is stored in the left-to-right order produced by FOREST so we can apply the numerical test at the end of Section I. For example, we can easily establish that the virtual nodes v(3, 10) and v(3, 11) of the forest of Fig. 4 are WHITE because they lie between (in left-to-right order) the successive elements (3, 9) and (4, 48) in the table. Since the elements of the table are linearly ordered by this left-to-right order, we may perform a binary search to check on the color of a virtual node. If T represents a picture in a $2^n \times 2^n$ grid, then such a search requires time

$$\begin{aligned} &O(\log(\text{number of trees in forest})) \\ &\quad = O(\log 4^{n-1}) \quad \text{(by the theorem)} \\ &\quad = O(n). \end{aligned}$$

For a real number x, let floor(x) denote the greatest integer less than or equal to x. The father of a virtual node v(L, K) is the virtual node v(L - 1, floor(K/4)). If P is a real node, then either its father is the real node given by FATHER(P) or else P is the root of a tree in the forest F. In the latter case, the coordinates, (L, K) of P, are recorded in the forest and we may deduce that P's father is v(L - 1, floor(K/4)). Obviously, if T is a quadtree, then R(F(T)) is identical to T.

The space savings for a forest of quadtrees can vary enormously. In the example of Figs. 2 and 4 there is a reduction from 41 nodes for the quadtree to 21 nodes. This may not be regarded as a 49 percent savings because there is a table with five entries. If we assume the space for an integer equals the space for a pointer and we disregard COLOR fields, then the space reduction is from (41)(5) = 205 pointers to (21)(5) + (5)(3) = 105 + 15 = 120 pointers, a savings of 41 percent. Greater space savings is associated with a larger table and, thus, greater degradation of time efficiency.

We have shown how the fundamental quadtree operations may be simulated on our virtual quadtrees. Also, we have considered the performance of these fundamental operations implemented on our representations shown that performance degradations are not severe. Since algorithms which work by moving about in a quadtree (finding neighbors, creating chain codes, determining pixel color, etc.) use the quadtree via the fundamental operations described, the algorithms may be implemented on our representations.

ACKNOWLEDGMENT

The authors wish to thank the referees for their hopeful comments. The compact quadtree modification of Fig. 3(b) was suggested by one of the referees. The need for an MTOEND field was noted by R. Zarling and the numerical test for left-to-right ordering was obtained independently by V. Raman.

REFERENCES

[1] N. Alexandridis and A. Klinger, "Picture decomposition, tree data-structures and identifying directional symmetries as node combinations," *Comput. Graphics Image Processing*, no. 8, pp. 43-47, 1978.

[2] R. C. Dyer, A. Rosenfeld, and H. Samet, "Region representation: Boundary codes from quadtrees," *Commun. Ass. Comput. Mach.*, vol. 23, pp. 170-179, Mar. 1980.

[3] 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.

[4] L. Jones and S. Iyengar, "Representation of a region as a forest of quadtrees," in *Proc. IEEE Conf. Pattern Recognition and Image Processing*, Aug. 1981, pp. 57-59.

[5] A. Klinger and C. R. Dyer, "Experiments on picture representation using regular decomposition," *Comput. Graphics Image Processing*, no. 5, pp. 68-105, Mar. 1976.

[6] D. E. Knuth, *The Art of Computer Programming, Vol. 1, Fundamental Algorithms.* Reading, MA: Addison-Wesley, 1969.

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

[8] —, "Connected component labeling using quadtrees," *J. Ass. Comput. Mach.*, vol. 28, pp. 487-501, July 1981.

[9] —, "An algorithm for converting rasters to quadtrees," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-3, pp. 93-95, Jan. 1981.

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

[11] T. Pavlidis, *Algorithms for Graphics and Image Processing.* Comput. Sci. Press, 1982.

[12] —, "The use of algorithms of piecewise approximations for picture processing applications," *ACM Trans. Math. Software*, vol. 2, pp. 305-321, Dec. 1976.

[13] S. L. Tanimoto and A. Klinger, Eds., *Image Data Structures: Structured Computer Vision.* New York: Academic, 1980.

Image Segmentation: A Comment on "Studies in Global and Local Histogram-Guided Relaxation Algorithms"

KEITH PRICE

The paper by Nagin *et al.*[1] neglects two important points. First there is no mention of an earlier histogram based segmentation method [1] which presents an alternative method to cope with the problem of overlaping intensity distributions. Second, the possibility of using a similar recursive technique is mentioned (in their flow chart on p. 263[1]) but the importance of this idea is ignored.

Recursive segmentation methods (e.g., [1], [2]) are much more powerful than applying the same technique at one level, without a great increase in algorithmic complexity. The strength of recursive segmentation methods is that decisions can be delayed until there is enough evidence. By selecting only the best

Manuscript received January 7, 1983. This work was supported by the Defense Advanced Projects Agency, the U.S. Department of Defense, and was monitored by the Wright-Patterson Air Force Base under Contract F-33615-80-(1080), DARPA Order 3119.

The author is with the Department of Electrical Engineering, University of Southern California, Los Angeles, CA 90089.

[1] P. A. Nagin, A. R. Hanson, and E. M. Riseman, *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-4, pp. 263-277, May 1982.