BIT 23 (1983), 472–486

PROPERTIES AND APPLICATIONS OF FORESTS OF QUADTREES FOR PICTORIAL DATA REPRESENTATION

VASUDEVAN RAMAN * and S. SITHARAMA IYENGAR **

* *Department of Electrical and Computer Engineering, Louisiana State Universiy, Baton Rouge, Louisiana 70803, USA*

** *Department of Computer Science, Louisiana State University, Baton Rouge, Lousiana 70803, USA*

Abstract.

Region representation as a quadtree data structure is a rich field in computer science with many different approaches. Forests of quadtrees offer space savings over regular quadtrees by concentrating the vital information [4, 5, 6]. They scavenge unused and unneeded space (i.e., node containing no information). This paper investigates several properties of forests of quadtrees which can be used to design manipulation algorithms for forest-quadtree data structure. In addition, the paper discusses the space saving and shows how the basic operations that can be performed on a quadtree can also be done on the more space efficient representation (a forest of quadtrees).

Keywords and phrases: quadtree. forest, data structure, image processing, algorithm.
CR Categories: 3.63, 8.2.

1. Introduction.

Efficient data structures for region representation are important for use in manipulating pictorial information. Recent research [1, 2, 3, 7, 8, 9] on quadtrees has produced several interesting results in different areas of image processing. A good tracing of the history of the evolution of quadtrees is provided by Klinger and Dyer [12]. Much work has been done on the quadtree properties and algorithms for manipulations and translations have been derived by Samet [9, 10], Dyer [1] and others [2, 5, 6]. For overviews of related research on image data structures see [4, 11, 12]. In 1981, methods of refining the quadtree were proposed by Jones and Iyengar [5]. The new refinements were called virtual quadtrees. Virtual quadtrees include both compact quadtrees and forests of quadtrees. The paper by Jones and Iyengar [4] further illustrates the usefulness of a forest of quadtrees as an efficient representation for binary images.

This paper is concerned with the properties of forests of quadtrees and their applications for picture processing and discusses development of forest manipulation algorithms.

Received February 19, 1983. Revised July 22, 1983.

Before the results are described, we begin by giving definitions and summary of previous results [4, 5, 6] in the next section of our paper.

2. Definitions and summary of previous results.

Pictures: A picture or raster is defined to be a grid of $2^n \times 2^n$ colored points (pixels), the color representing properties associated with the points.

Quadtrees: A quadtree is a tree structure with the restriction that any node must have either four offspring (or children or descendents) or none.

Quadtrees for pictorial representation. In a quadtree representing a picture, the root represents the whole picture. Its offspring represent each one quadrant in the order Northwest(NW), Northeast(NE), Southwest(SW), and Southeast(SE). These four children are numbered from 0 to 3. In turn, their offspring each represents a subquadrant of the four quadrants and so on until the maximum number of subdivisions have been made as determined by the resolution of the image. In addition, if the children of a node are all the same color, they are deleted and their parent receives the information that was common to the four children. They are simply not needed as they carry redundant information. Figures 1a and 1b show a typical picture of a simple region and its quadtree

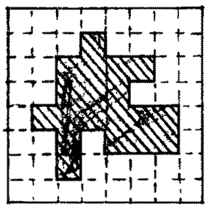


Fig. 1(a). Sample region.

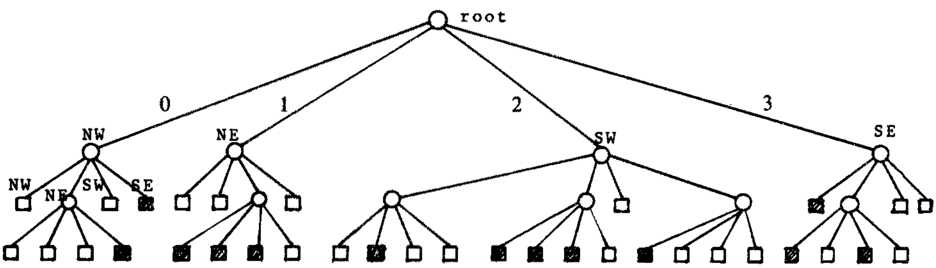


Fig. 1(b). Quadtree for sample region shown in Fig. 1(a) representation. In this quadtree, parents hold the information "GRAY" and leaves are either "BLACK" or "WHITE", representing the presence or absence of color respectively.

Node in a quadtree. Nodes in the quadtree are required to hold color information and the 4 pointers to their children. In addition, in the forest transformation an additional datum is needed, the "TYPE" field. A typical node in a quadtree, then, appears in storage as shown in Figure 2.

<table>
<tr><td colspan="2">COLOR</td><td colspan="2">TYPE</td></tr>
<tr><td>NW</td><td>NE</td><td>SW</td><td>SE</td></tr>
</table>

Fig. 2. Node of a quadtree.

When additional manipulation is done, such as roping of the quadtree, more fields may be added.

In our quadtrees, the COLOR field holds either "BLACK", "WHITE", or "GRAY", the TYPE field "GOOD" or "BAD", and the other fields are pointers holding either all nulls or all pointer values (addresses to subtrees).

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.

For a broader treatment on this, see [4, 6].

Forest of quadtrees. Let T be a quadtree. The quadtree T is represented by forest, $F(T)$, of quadtrees consisting of a list of triples of the form (P, L, K) and a collection of quadtrees where

(a) each triple (P, L, K) in the list consists of the coordinates, (L, K), of a node in T, and a pointer, P, to a quadtree in the collection isomorphic to the subtree rooted at position (L, K) in T;
(b) if (L, K) and (M, N) are coordinates of nodes recorded in F, then neither node is the root of a subtree containing the other;
(c) every BLACK leaf in T is represented by a leaf in $F(T)$.

For example, Figure 3 contains a forest that represents the quadtree of Figure 1b. The idea of the algorithms for reducing a tree T to a forest of quadtrees $F(T)$ that represents the tree, can be described informally as follows.

First, a "labelling" algorithm is executed. This algorithm traverses the quadtrees depth-first and labels each node "GOOD" and "BAD", depending

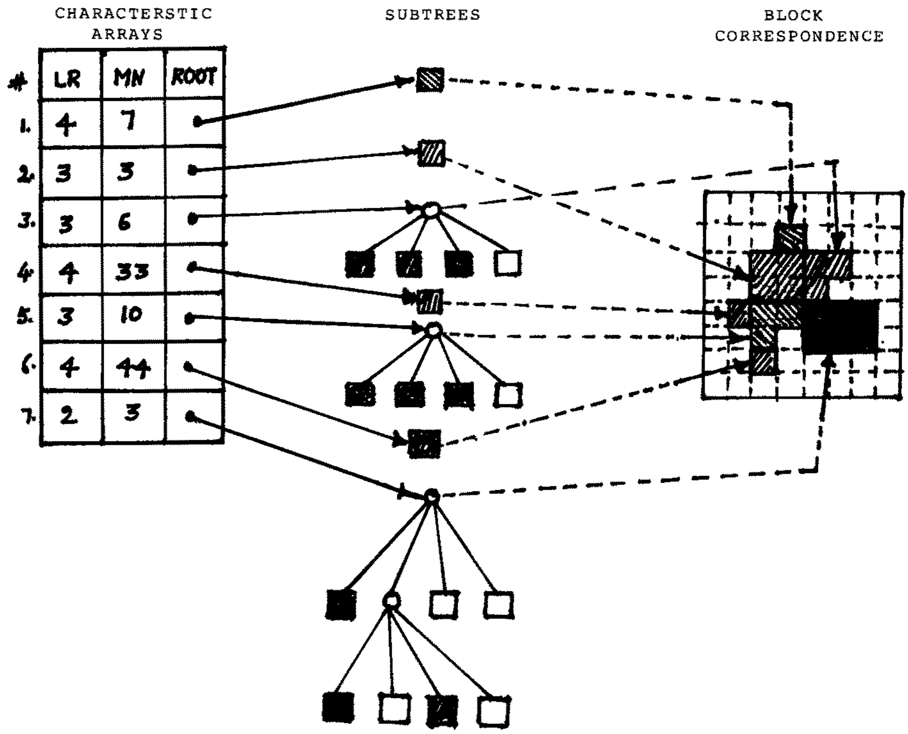


Fig. 3. Forest of quadtrees for sample region shown in Fig. 1(a). The block correspondence of the region shown in Fig. 1(a).

upon its importance in information storage. A GOOD node is either a "BLACK" leaf or a parent of two or more GOOD children. All other nodes are labeled "BAD".

The second phase is a forest creation algorithm. This algorithm retraverses the quadtree, deleting BAD nodes from the top down until it encounters a GOOD node. This node now becomes the root of a subtree in the forest, and its corresponding triple (i.e., level, magic number and pointer to its root) is stored in the characteristic arrays. If this node is a "BLACK" leaf, it becomes a "BLACK" leaf in the forest, which is actually a subtree consisting of but one node.

The path code. If the children of a node are numbered so that 0 represents NW, 1 represents NE, 2 represents SW, and 3 represents SE, then a path from the root to another node on level L in a quadtree can be represented by a quaternary number with $L-1$ digits (the root has level number 1). For instance, the quaternary number denoting the path from the root to the leaf marked X on figure 1b has the value 201 (which is 33 decimal) since the directions taken from the root to X are SW, NW, and NE. This number is called the path code and forms the K coordinate in the triple representing a subtree in a forest.

If the path code for a node on level L is M_L, then the path code of a child is

$$M_{L+1} = 4M_L + D \tag{1}$$

where D is the digit representing the chosen direction. This follows directly from the definition of the path code.

In [4] the following theorem was proved:

THEOREM 1. *The maximum number of trees in a forest derived from a quadtree that represents a square of dimensions* $2^K \times 2^K$ *is* 4^{K-1}, *which is one-fourth the area of the square.*

Having reviewed the definitions and results of previous work, we now present the new results on the structural properties of forests of quadtrees.

3. Structural properties of forests of quadtrees.

The following theorems and their corollaries are useful in the development of forest manipulation algorithms.

THEOREM 2. *Given a node with level* L_F *and path code* M_F, *any arbitrary node with level* $L_N > L_F$ *and path code* M_N *can be a descendent only if the following inequality holds*:

$$M_F \times 4^{L_N - L_F} \leqq M_N < (M_F + 1) \times 4^{L_N - L_F}.$$

PROOF. By always taking the minimum value of D in (1), we obtain the following recurrence relation:

$$M_{L+1} = 4M_L \qquad (M_L = M_F \text{ for } L = L_F)$$

and by always taking the maximum D we have:

$$M_{L+1} = 4M_L + 3 \qquad (M_L = M_F \text{ for } L = L_F)$$

The first equation is homogeneous and has the solution:

$$M_L = M_F \times 4^{L - L_F}. \tag{2}$$

A particular solution to the second equation is:

$$M_L = 4^{L-L_F}-1$$

so the solution is:

$$M_L = (M_F+1)4^{L-L_F}-1. \tag{3}$$

Since M_N must lie between the two values given by (2) and (3) the theorem is proved. ■

COROLLARY 1. *Given a node with level L_F and path code M_F, an arbitrary node with level $L_N < L_F$ and path code M_N can be ancestor if and only if the following inequality holds:*

$$M_N \times 4^{L_F-L_N} \leqq M_F < (M_N+1) \times 4^{L_F-L_N}. \tag{4}$$

PROOF. If a node N is an ancestor of a node F, then node F must be a descendant of node N. ■

COROLLARY 2. *Given a node with level L_F and path code M_F, an arbitrary node with level $L_N \leqq L_F$ and path code M_N can be an uncle of the node (the child of an ancestor) if and only if the following inequality holds:*

$$4 \times \lfloor M_F/4^{L_F-L_N+1} \rfloor \leqq M_N \leqq 4 \times \lfloor M_F/4^{L_F-L_N+1} \rfloor + 3.$$

PROOF. The father of node F (L_F, M_F) is

$$(L_F-1, \lfloor M_F/4 \rfloor).$$

Since for all $x > 0$, $\lfloor \lfloor x/4 \rfloor /4 \rfloor = \lfloor x/4^2 \rfloor$, an ancestor on level L_N-1 is

$$(L_{\text{ANC}}, M_{\text{ANC}}) = (L_N-1, \lfloor M_F/4^{L_F-L_N+1} \rfloor).$$

Three of the children to this node are uncles for node F and the fourth is an ancestor. But this means that an uncle's path code must lie in the range (cf. formula (1))

$$4M_{\text{ANC}}+0 \leqq M \leqq 4M_{\text{ANC}}+3$$

which proves the corollary. ■

COROLLARY 3. *Given a subtree with root R (L_R, M_R), an arbitrary node N (L_N, M_N), where $L_N \geqq L_R$, is situated in an adjacent subtree*
--- to the left if:

$$M_N < M_R \cdot 4^{L_N - L_R}$$

--- *to the right is*:

$$M_N \geqq (M_R + 1)4^{L_N - L_R}.$$

PROOF. Since the path codes of nodes on the same level L in a quadtree increase towards the right from 0 to $4^{L-1} - 1$, theorem 2 shows that an upper bound of path codes for subtrees to the left of node R is given by:

$$M_N < (M_R - 1 + 1)4^{L_N - L_R}$$

which reduces to the following inequality

$$M_N < M_R \cdot 4^{L_N - L_R}$$

while a lower bound of the path codes for subtrees to the right of node R is given by:

$$M_N \geqq (M_R + 1)4^{L_N - L_R}. \qquad \blacksquare$$

THEOREM 3. *Given two nodes with levels L_A and L_B and path codes M_A and M_B and a common ancestor with level L_{ANC} and path code M_{ANC}, the shortest path between the two nodes (in edges) is*:

$$d_{\text{SP}} = (L_A + L_B) - 2L_{\text{ANC}}.$$

PROOF. It is evident upon examination of a quadtree or subtree of a forest that the distance of the shortest path between any two nodes is the sum of the distances to a common ancestor from each one. This is simply the expression:

$$(L_A - L_{\text{ANC}}) + (L_B - L_{\text{ANC}}). \qquad \blacksquare$$

4. Algorithms.

Preliminary manipulation algorithms include traversal, search, and the reverse of forest creation, reconstruction. Due to the complexity of a hypothetical traversal algorithm, at least to produce results equatable with those of a quadtree traversal algorithm, it was decided to forego this less useful manipulation for the more useful search and reconstruction algorithms. These are now presented.

The search algorithm, entitled "FSEARCH" searches for a node in a forest by coordinates, and upon the location of the node, subsequently returns its pointer if it is real, or its color if it is virtual.

The input to the process is a forest of quadtrees in pointer-based storage, three arrays, ROOT(n), $LR(n)$, and $MN(n)$, each holding the nth subtree root pointer, level, and path code, respectively, the number of subtrees N in the forest, and finally the level L and magic number M of the node to be searched for.

The program works by performing a binary search on the list of subtrees until a subtree is located whose root has the wanted node as an ancestor, descendant, or "uncle". The node is then located and characterized.

There is no output except the possible error message. A pointer and the ancestor flag AF are returned.

The reconstruction algorithm, entitled "RECONS", reverses the creation of the forest.

The algorithm works by creating all ancestors of the first subtree and their tentative descendants (white nodes). Then each subtree thereafter is connected, via a chain of ancestors, to a common ancestor with the first subtree.

The input to the algorithm is the same as that of "FSEARCH", except that only N, the number of subtrees in the forest, is needed in addition to the forest and its characteristic arrays. Also, the algorithm assumes existence of a zero element in the LR and MN arrays initialized to zero ($LR(0) = 0$, $MN(0) = 0$).

There is no output and Q, the quadtree root pointer, is the only thing returned. We shall now present the algorithm in a pascal like syntax.

FSEARCH:

procedure FSEARCH (l, m, n: **integer**; p: **ptr**; af: **boolean**);

This procedure searches for a given node with level l and path code m. The number of subtrees is given as n, t is the subtree index, s and b determine t according to binary search rules, and d is the horizontal distance between nodes. The first condition tests to see if t has gone out of the list of subtrees, the second tests for descendants, the third for ancestors, the fourth for uncles, the fifth for virtual white nodes, and the sixth and seventh for direction to search for the node. The last condition is a blatant error in the forest. The af is true for a gray ancestor found and p is null for a virtual node, and the pointer to a real node.

```
begin
  s := 1;
  b := n;
  t := trunc((s+b)/2);
  d := 0;
  while true do
    begin
```

```
if t < 1 or t > n then do
  begin
    af := false;
    p := null;
    return;
  end;
else if (mn(t)*4**(l-lr(t)) ≦ m) and (m < (mn(t)+1)*4**(l-lr(t))) and
  l ≧ lr(t) then do
    begin
      af := false;
      p := travers(l, m, root(t));
      return;
    end;
else if (m*4**(lr(t)-1) ≦ mn(t)) and (mn(t) < (m+1)*4**(lr(t)-l)) then
     do
    begin
      af := true;
      p := null;
      return;
    end;
else if (trunc(mn(t)/4**(lr(t)-l+1))*4 ≦ m) and
  (m ≦ trunc(mn(t)/4**(lr(t)-l+1)*4+3) then do
    begin
      if d = 0 then do
        begin
          d := m-trunc(mn(t)/4**(lr(t)-l));
          t := t+d/abs(d);
        end;
      else t := t+d/abs(d);
    end;
else if d < > 0 then do
  begin
    af := false;
    p := null;
    return;
  end;
else if m < mn(t)*4**(l-lr(t)) then do
  begin
    b := t-1;
    if s > b then do
      begin
        af := false;
        p := null;
```

```
          return;
        end;
      t := trunc((s+b)/2);
    end;
  else if m ≧ (mn(t)+1)*4**(l-lr(t)) then do
    begin
      s := t+1;
      if s > b then do
        begin
          af := false;
          p := null;
          return;
        end;
      t := trunc((s+b)/2);
  else do
    begin
      writeln('*** severe fatal error - bad forest');
      stop;
    end;
  end;
end;
end FSEARCH;
```

RECONS:

procedure RECONS (n: **integer**, q: **ptr**);

This procedure reconstructs a normal quadtree from a forest of quadtrees, returning q as the new quadtree root pointer. Each subtree is indexed in turn by t and the ancestors allocated and linked upward as shown in figure 4(a) and 4(b), the former representing the first iteration, and the latter representing a possible later iteration. The variable m is the new ancestor's path code and d is the horizontal path code distance. The main if-stmt. checks to see whether the subtree and its ancestors should be connected to the left adjacent subtree and ancestors of that subtree. Pointers $p1$, $p2$, $p3$ point to white nodes which serve to provide the ancestor with four proper descendants. Pointer p points to the new ancestor.

```
begin
  for t := 1 to n do
    begin
      h = root(t);
      for l := lr(t)-1 downto 1 do
        begin
          m = trunc(mn(t)/4**(lr(t)-l));
```

```
d = m-trunc(mn(t)/4**(lr(t)-l)+1))*4;
if t = 1 or not ((m*4**(lr(t-1)-l) ≦ mn(t-1)) and (mn(t-1) <
  (m+1)*4**(lr(t-1)-l))) then do
    begin
      new(p);
      if l = 1 then q = p;
      new(p1);
      new(p2);
      new(p3);
      p1.color := 'white';
      p1.nw := null;
      p1.ne := null;
      p1.sw := null;
      p1.se := null;
      p2.color := 'white';
      p2.nw := null;
      p2.ne := null;
      p2.sw := null;
      p2.se := null;
      p3.color:='white';
      p3.nw:=null;
      p3.ne:=null;
      p3.sw:=null;
      p3.se:=null;
      if d=0 then do
        begin
          p.nw:=h;
          p.ne:=p1;
          p.sw:=p2;
          p.se:=p3;
        end;
      else if d=1 then do
        begin
          p,nw:=p1;
          p.ne:=h;
          p.sw:=p2;
          p.se:=p3;
        end;
      else if d=2 then do
        begin
          p.nw:=p1;
          p.ne:=p2;
          p.sw:=h;
```

```
        p.se: =p3;
      end;
        else if d=3 then do
          begin
            p.nw:=p1;
            p.ne:=p2;
            p.sw:=p3;
            p.se:=h;
          end;
        else do
          begin
            writeln('*** error in reconstruction ***');
            stop;
          end;
        p.color:='gray';
        h:=p;
      end;
  else do
    begin
      p:= travers(l,m,q);
      if d=0 then p2:=p.nw;
      else if d=1 then p2:=p.ne;
      else if d=2 then p2:=p.sw;
      else if d=3 then p2:=p.se;
      if p2.color< >'white' then do
        begin
          writeln('*** ERROR- BAD FOREST ***');
          stop;
        end;
      if d=0 then do
        begin
          dispose(p.nw);
          p.nw:=h;
        end;
      else if d=1 then do
        begin
          dispose(p.ne);
          p.ne:=h;
        end;
      else if d=2 then do
        begin
          dispose(p.sw);
          p.sw:=h;
```

```
                end;
              else if d = 3 then do
                begin
                  dispose(p.se);
                  p.se = h;
                end;
              exit for;
            end;
        end;
      end;
  end;
end RECONS;
```

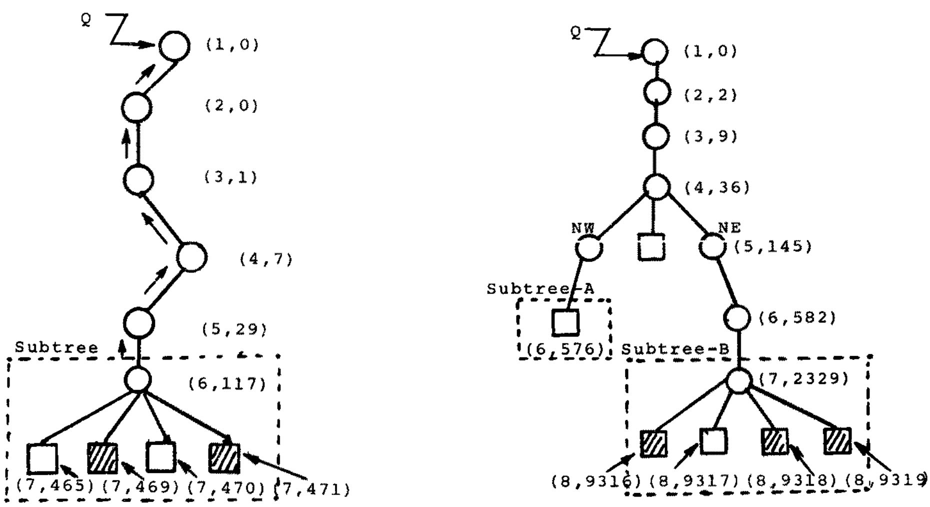
5. Discussion of results.

Having presented the properties and algorithms of forests, we now state our conclusions from the work done so far.

Theorem 2 and its corollaries were used in the development of the manipulation algorithms for a forest of quadtrees.

Theorem 3 can be used to find the distance in edges between two nodes in a quadtree provided we know the level of a common ancestor of both nodes.

In this paper, we have tested both algorithms, finding the results very good in terms of time and space efficiency. The algorithms were tested in several different programming languages. The complexity of both algorithms is linear in the



(a) Fig. 4. Constructing ancestor chain up. (b) number of subtrees present, which is related to the size and resolution of the image. Empirical results concerning the relative time and space usage of the forest of quadtrees and an even newer structure developed by the authors versus a normal quadtree are reported in [6].

The newer structure, called a hybrid quadtree, utilizes not only the method of the forest of quadtrees, but also draws from the "metanode" format of the compact quadtree and upon further techniques. The integrity of the quadtree is preserved. For more of this see [15].

6. Summary and conclusions.

The forest data structure seems to offer general benefits over the normal quadtree as the worst case could be where both structures are equal. Even if the additional space taken by the characteristic arrays is considered, the benefits of using a forest of quadtrees can still be realized. Recent work by the authors has yielded an even newer structure called a hybrid quadtree [15] which combines the advantages of the forest of quadtrees and the compact quadtree, with a resulting much higher space and time efficiency. This is very significant and research is still in progress.

In previous work by Iyengar and Jones [4, 5] and in the recent publication by Gargantini [13], other possible structures to provide space-savings over the quadtrees were proposed. The primary advantage of forests over other possible structures appears to be its ease of use stemming from its similarity and compatibility with the widely-used quadtree structure used today. This is something that must be considered in any pictorial application of a data structure.

Further work needs to be done in the manner of advanced manipulation of forests, such as roping, and more algorithms to equate the forest with normal quadtrees in ease of application.

Acknowledgements.

We would like to thank Professor Jones and Dr. Madhavan for helpful discussions of this work. We are indebted to Professor Venkatesh and the referees for their critical comments on an earlier version of this paper. We are very grateful to one referee for several important suggestions on various theorems in our paper. We would also like to thank the referee for his implementation in SIMULA.

REFERENCES

1. C. R. Dyer, A. Rosenfeld and H. Samet, *Region representation: Boundary codes from quadtrees*, Comm. ACM 23, 3 (March 1980), 171–179.
2. G. M. Hunter and K. Steiglitz, *Operations on images using quadtrees*, IEEE Trans. On Pattern Analysis and Intell. 1, 2 (April 1979), 145–153.
3. G. M. Hunter and K. Steiglitz, *Linear transformations of pictures represented by quadtrees*, Computer, Graphics and Image Processing 10, 3 (July 1979), 289–296.
4. L. Jones and S. S. Iyengar, *Space and time efficient virtual quadtrees*, Technical Report #82-D24, Louisiana State University, Baton Rouge, LA 70803, (Nov, 1982).
5. L. Jones and S. S. Iyengar, *Representation of a region as a forest of quadtrees*, Proc. IEEE-PRIP 81 Conference, Dallas, TX, IEEE Publ. 81 Ch1595-8, (1981), 57–59.
6. L. Jones and S. S. Iyengar, *Virtual quadtrees*, Proc. IEEE-CVIP 83 Conference, Virginia, IEEE publications (1983), 101–105.
7. E. Kawaguchi and T. Endo, *On a method of binary-picture representation and its application to data compression*, IEEE Trans. On Pattern Analysis and Machine Intell. 2, 1 (Jan. 1980), 27–35.
8. A. Klinger, *Patterns and search statistics, optimizing methods in statistics*, J. D. Rustagi, (Ed.), Academic Press, New York (1971).
9. A. Klinger and M. L. Rhodes, *Organization and access of image data by areas*, IEEE Trans. On Pattern Analysis and Machine Intell. 1, 11 (Jan. 1979), 50–60.
10. H. Samet, *An algorithm for converting rasters to quadtrees*, IEEE Trans. On Pattern Analysis and Machine Intell. 3, 1 (Jan. 1981).
11. H. Samet, *Connected component labelling using quadtrees*, JACM 28, 3 (July 1981), 487-501.
12. H. Samet, *Region representation: Quadtrees from boundary codes*, Comm. ACM 23, 3 (March 1980), 163–170.
13. S. Klinger and C. R. Dyer, *Experiments on picture representation using regular decomposition*, Computer Graphics and Image Processing, No. 5 (March 1976), 68–105.
14. Irene Gargantini, *An effective way to represent quadtrees*, Comm. of ACM, 25, 12 (Dec. 1982), 905–910.
15. V. Raman, S. S. Iyengar and S. K. Kandu, *An optimal quadtree structure for pictorial data representation using top and bottom compaction technique*; to appear in the Proceedings of IEEE-SMC Conference, Bombay, India (Dec. 1983).