# Concise Papers

## Threaded Linear Hierarchical Quadtrees for Computation of Geometric Properties of Binary Images

A. UNNIKRISHNAN, PRITI SHANKAR, AND Y. V. VENKATESH

*Abstract*—**A modification of the linear quadtree [3], the threaded linear hierarchical quadtree (TLHQT), is proposed for the computation of geometric properties of binary images. Since most of the algorithms used in connection with computation of geometric properties require frequent exploration of adjacencies, a structure which keeps permanently in memory some adjacency links is introduced. In this paper, we present some results obtained by using the TLHQT for labeling connected components, evaluating perimeter and Euler's number in a quadtree environment. The algorithms for computing perimeter and Euler number and the first phase of the labeling algorithm are shown to have time complexity $O(B)$, where $B$ is the number of black nodes of the quadtree. The authors determine the adjacency links at the very beginning—namely, when the binary image is mapped from raster scan to the quadtree. Pixel adjacency is, in fact, available during row scanning, and node's adjacency is easy to evaluate locally when performing condensation of nodes into larger quadrants and also while merging partial quadtrees. Although the structure requires space nearly four times as much as the linear quadtree, the requirement is roughly half that for the pointer-based quadtree. Also it appears that for computing geometric properties, the TLHQT offers execution timings better than those obtained by both the linear and pointer-based quadtrees and the graph structure reported in [16].**

*Index Terms*—**Binary images, geometric properties, hierarchical quadtree, linear quadtree, quadtree.**

### INTRODUCTION

It is known that, among the various data structures proposed for representing binary images, quadtrees lead to a saving in storage and facilitate the implementation of many operations on these images, like the computation of geometric properties. The quadtree arises from the representation of a regular decomposition of a square grid of size $2^n \times 2^n$, enclosing the image, into homogeneously colored quadrants, the smallest quadrant being a pixel [1].

There are two commonly used representations of the quadtree in the computer:

1) Each node of the tree is represented as a record with six fields, in which four fields are pointers to the sons, one is a pointer to the father and the last encodes the color (i.e., black, white, or gray) of the node [1], [2], [7]–[9].

2) Each black node is represented as an $n$-digit quaternary code ("$q$-code") and the codes are numerically ordered in an array to get a linear quadtree, thereby avoiding pointers.

Fig. 1(a)–(d) illustrates the two representation schemes. Other representation schemes include the forest of quadtrees [13], the DF expression [14], the leaf codes [15], the $B^+$-trees [17], and more recently the graph quadtrees [16]. For a detailed review of the various representation schemes, see [2].

Manuscript received February 27, 1987; revised October 30, 1987.

A. Unnikrishnan and Y. V. Venkatesh are with the Department of Electrical Engineering, Indian Institute of Science, Bangalore 560 012, India.

P. Shankar is with the Department of Computer Science and Automation, Indian Institute of Science, Bangalore 560 012, India.

IEEE Log Number 8819888.

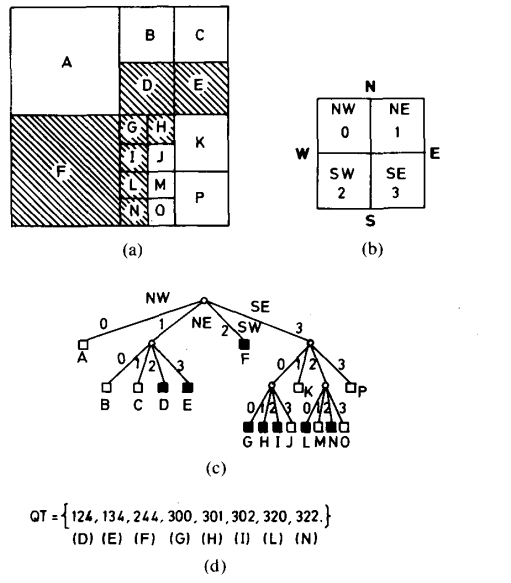



Fig. 1. Quadtree for a binary picture. (a) $2^3 \times 2^3$ binary picture. (b) Assignment of links. (c) Pointer based quadtree. (d) Linear quadtree (4 is used as label).

In this paper, we introduce a modification of the linear quadtree called the threaded linear hierarchical quadtree (TLHQT) and explore the possibility of deriving this structure directly from the raster scan of a binary image, to be used later to compute the geometric properties of binary images. We consider the following three typical geometric properties:

1) connected component labeling
2) perimeter
3) Euler number.

The rest of the paper is organized as follows.

*Section II:* Threaded linear hierarchical quadtree (TLHQT).

*Section III:* Conversion of a raster-scanned binary image to TLHQT.

*Section IV:* Computation of geometric properties.

*Section V:* Concluding remarks.

### II. THREADED LINEAR HIERARCHICAL QUADTREE (TLHQT)

The computation of most of the geometric properties of binary images needs information regarding spatially adjacent nodes of the quadtree. In the pointer-based structure, traversal of the appropriate links is necessary to locate a neighbor at same level [4] while in the linear quadtree, this is achieved by first computing the $q$-code of a neighbor of same size and then searching for the presence of the neighbor $q$-code in the linear quadtree itself [3]. The authors [6] have shown that by structuring the $q$-codes (in terms of size) into a hierarchy of arrays, the search for a $q$-code of size $4^k$ is restricted to the subarray at level $k$. Also, by using a bottom-up approach, which analyses, from the pixel level $q$-codes, in size-increasing order, the computational time can be further reduced.

In this paper, we propose a structure to eliminate these searches altogether. We do this by generating and storing in each node the information about neighbors during the raster scan conversion, in

TABLE I
THE LHQT FOR THE BINARY IMAGE OF FIG. 1(a)

| Level | Hierarchically Ordered $q$-Code. | Linear Hierarchical $q$-Codes (LHQC) |
|---|---|---|
| 2 | 244 | 2 |
| 1 | 124 | 12 |
| | 134 | 13 |
| 0 | 300 | 300 |
| | 301 | 301 |
| | 302 | 302 |
| | 320 | 320 |
| | 322 | 322 |

TABLE II
THREADED LHQT FOR THE IMAGE OF FIG. 1(a)

| level | index | LHQC | THREAD as a linked list |
|---|---|---|---|
| 2 | 1 | 2 | NIL |
| 1 | 2 | 12 | E 3 |
| | 3 | 13 | NIL |
| 0 | 4 | 300 | N 2 → E 5 → S 6 → W 1 |
| | 5 | 301 | N 2 |
| | 6 | 302 | S 7 → W 1 |
| | 7 | 320 | S 8 → W 1 |
| | 8 | 322 | W 1 |

addition to the $q$-code of the node. The main idea is to link a black node, identified by a $q$-code and its position in one of the $n$ arrays, with its black neighbors at the same or higher level of hierarchy. The threading is thus different from the one proposed by Hunter and Steiglitz [5]. In the "roped" quadtrees proposed by Hunter and Steiglitz, every black node has pointers (or ropes) to its four neighbors. But the neighbors pointed to need not be all leaf nodes, with the result additional tree traversals are necessary to locate the neighboring leaf nodes. It can be seen that in the new structure called the threaded linear hierarchical quadtree (TLHQT), every node pointed to is a black node. The computation of geometric properties involving exploration of adjacencies is therefore faster than using a "roped" quadtree.

We now give the definition of TLHQT, based on the LHQT proposed by the authors [6]. Note that if only the black nodes of a pointer-based quadtree (or the $q$-codes of a linear quadtree) are arranged in a hierarchy of $n$ arrays, according to size, we obtain the LHQT with the following properties.

1) The array at level $k (0 < k < n - 1)$ contains $b_k$ black nodes of size $4^k$, represented by $(n - k)$ quaternary digits called linear hierarchical $q$-codes (LHQC's).

2) The LHQC's in each level $k$ are sorted in an increasing order.

If, in addition, links to the neighbor nodes along the directions NORTH, EAST, SOUTH, and WEST at the same or higher levels (i.e., at levels $> k$) of hierarchy are provided, the threaded linear hierarchical quadtree (TLHQT) is obtained. Formally, the TLHQT is defined as follows (using Pascal-like constructs):

```
type  INDEX = 1 .. MAX; {MAX is a large integer}
      DIRECTION     = (NORTH, EAST, SOUTH, WEST);
      THREADNODE = record
                     LINK : INDEX;
                     DIR : DIRECTION;
                     NEXT : ^ THREADNODE;
                   end;
      QNODE        = record
                     QCD : LHQC;
                   THREAD : ^ THREADNODE;
                   end;
      TLHQT        = array [INDEX] of QNODE;
var   T    : TLHQT;
```

*Remark 1:* If $T[I]$.thread = nil then $T[I]$ has no black neighbor at the same or higher levels.

*Remark 2:* A QNODE at level $k$ will have links to a neighbor of same size only along the directions EAST and SOUTH, as these QNODE's are sorted in increasing order, with respect to the LHQC of the QNODE.

For illustration, Tables I and II present, respectively, the LHQT and the TLHQT for Fig. 1(a).

## III. CONVERSION OF A RASTER SCANNED BINARY IMAGE TO TLHQT

Often, the input to an imaging system is derived from a raster scan of the scene, and, therefore, it is desirable to build the TLHQT as and when the raster rows are scanned. In [11], an algorithm is given to construct the LHQT from a raster scan. Other algorithms for building quadtrees from raster can be found in [18], [19]. In this paper, we show that the TLHQT can also be similarly constructed, using a marginal overhead of space and time (when compared to the generation of LHQT), required to introduce the threads.

The basic idea is to build the TLHQT from the top and bottom halves of the image separately. This generates two sets of linked lists, one set of each half, storing the QNODE's, as defined in Section II. Thus if LIST_TOP[k] and LIST_BOT[k] are the two sets of linked lists for $k = 0, 1, \cdots n - 2$, the two are merged to produce a final hierarchy of $n$ lists, representing the TLHQT for the image. The TLHQT for each half is generated by recursively subdividing each half into subhalves and combining the partial TLHQT's generated for each subhalf.

As an illustration of the procedure, consider the $2^3 \times 2^3$ image of Fig. 2(a), with rows numbered from 0 to 7. The partial TLHQT for rows 0, 1 is first formed and stored as LIST_TOP[$k$], $k = 0$, 1. Next, the partial LHQT for rows 2 and 3 is constructed and stored as LIST_BOT[$k$], $k = 0$, 1. Then LIST_TOP[$k$] and LIST_BOT[$k$], $k = 0$, 1, are combined (i.e., condensed and merged) to yield LIST[$k$], $k = 0$, 1, 2. LIST[$k$] will be the LIST_TOP[$k$] for rows 0..3. Similarly, LIST_BOT[$k$], $k = 0$, 1, 2 is generated for rows 4..7. LIST_TOP[$k$] and LIST_BOT[$k$], $k = 0$, 1, 2, 3 are combined to yield the final TLHQT in the linked list, LIST.

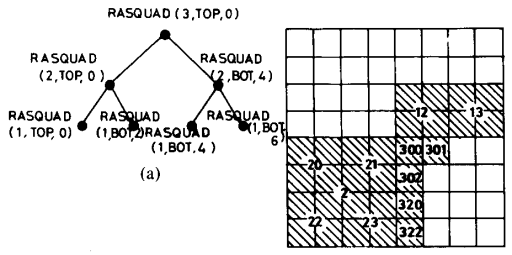
The above technique is implemented using a recursive procedure RASQUAD($k$, $l$, lineno). This procedure is initially invoked with parameter values $k = n$, $l$ = top and lineno = 0. The procedure is recursively invoked by decrementing $k$ by 1 until $k = 1$. The call sequences of the procedure, when applied to the binary image of Fig. 2(a) are given in Fig. 2(b). For $k = 1$, the first two rows (numbered 0 and 1) of the image are read into a buffer and the corresponding TLHQT, viz. LIST_TOP[$k$], $k = 0$, 1, for these two rows is constructed. Next, for the second instance of $k = 1$, rows 2 and 3 are read into a buffer and the TLHQT, LIST_BOT[$k$], $k = 0$, 1, is constructed. Then LIST_TOP and LIST_BOT are combined to form the TLHQT, LIST [$k$], $k = 0$, 1, 2, for rows 0..3. This operation is repeated until the TLHQTs for the top and bottom halves of the image are combined.

The complete algorithm consists of two distinct stages:

1) Generating the partial TLHQT from the scan rows corresponding to $k = 1$.

2) Combining the TLHQT's of the top and bottom halves of the portion scanned, corresponding to $k > 1$, to produce a single list, LIST [$i$], $i = k, k - 1, \cdots 1, 0$.

The latter stage involves two steps: consider level $k > 1$ of recursion.




RASQUAD (1,TOP, 0) generates (1) CAND_TOP= NIL
(2) LIST_TOP[1] = NIL , LIST_TOP [0]=NIL
RASQUAD(1,BOT, 2) generates (1) CAND_BOT =( 12 , 13 ) .
(2) LIST_BOT[1]=(12,13), LIST_BOT [0] = NIL
RASQUAD(2,TOP, 0) generates (1) CAND_TOP =(12 ,13 )
(2) LIST_TOP[2]= NIL , LIST_TOP[1] = ( 12, 13 )
LIST_TOP[0] = NIL
RASQUAD (1,TOP, 4 ) generates (1) CAND_TOP = (302)
(2) LIST_TOP[1]=(20,21 ), LIST_TOP[0]
= ( 300, 301, 302 )
RASQUAD (1,BOT, 6) generates (1) CAND_BOT = ( 320 )
(2) LIST_BOT [1] =(22, 23 )
LIST_BOT [0] = ( 320 , 322 )
RASQUAD (2 , BOT, 4) generates (1) CAND_BOT = (2,300 , 301 )
(2) LIST_BOT [2] = ( 2)
LIST_ BOT [1] = NIL
LIST_ BOT [0] = ( 300 , 301, 302, 320, 322 )
RASQUAD ( 3,TOP, 0 ) generates (1) LIST [2] =(2) , LIST [1] = ( 12 , 13 )
LIST [0] = ( 300, 301, 302 320, 322 )

(c)

Fig. 2. Execution of the procedure RASQUAD for a typical image. (a) Recursive invocation of RASQUAD. (b) The 8 × 8 image of Fig. 1(a). (c) Execution of the procedure RASQUAD for a typical image.

a) If four $q$-codes of size $4^{k-1}$, two from LIST_TOP[$k-1$] and two from LIST_BOT[$k-1$], happen to be in a quadrant, they "condense" to a single $q$-code of size $4^k$, which gets entered into LIST[k].

b) The individually sorted lists of LIST_TOP[$i$] and LIST_BOT[$i$] are pairwise merged for $i = k-1, k-2, \cdots, 0$ into a single list LIST[$i$].

The algorithm outputs an array of lists, LIST[$k$], $k = 0, 1, \cdots, n-1$, where LIST[$k$] contains the threaded QNODE's at level $k$. LIST is easily converted into the array TLHQT (defined in Section II). See Fig. 2 for an illustration of the threading, condensation, and merging operations.

Threads are introduced at three stages as explained below. Let the EAST and WEST threads be called horizontal threads and the NORTH and SOUTH threads, the vertical threads.

*Stage a):* During the scanning of a pair of rows, the pixels are read in quadrant-wise. Thus the vertical threads (across pixels) and the horizontal threads are easily introduced by computing the adjacencies with the last three QNODE's generated.

*Stage b):* When four QNODE's of level $k-1$ condense to a QNODE Q1 at level $k$, the horizontal threads to the previously generated QNODE Q2 of level $k$ are introduced when Q1 is inserted into LIST[$k$]. All QNODE's of level $< k-1$ that were previously linked to any four of the condensing QNODE's must be relinked to Q1. It can be seen that such QNODE's to be relinked are only horizontal to Q1.

*Stage c):* Vertical threads are introduced before merging LIST_TOP[$i$] and LIST_BOT[$i$], $i = k-1, k-2, \cdots, 0$. At level $k$, when RASQUAD returns LIST_TOP and LIST_BOT, it also returns two additional lists CAND_TOP and CAND_BOT, respectively, for the top and bottom halves of the section being scanned, which contain the QNODEs of level $< k-1$ sharing a common boundary across the two halves. The vertical threads are introduced across the elements of CAND_TOP and CAND_BOT.

The space and time requirements of the algorithm can be shown to be same as that required for generating LHQT plus an extra amount of space and time required for the lists CAND generated at various instances. (It may be noted that when the procedure RASQUAD returns, the list CAND is returned as CAND_TOP or CAND_BOT.) In the Appendix, it is shown that in the case of images whose levelwise distribution of LHQCs is skewed towards pixels, the additional space requirement is $O(Bn^2/2^n)$ and the corresponding time requirement is $O(B)$. Table III gives the total execution timings and extra space consumed by the algorithm for six typical cases of a 64 * 64 image. The time required to build LHQT, given in Table III, shows that the extra amount of time to build TLHQT is of $O(B)$.

TABLE III
COMPARISON OF EXECUTION TIMINGS (IN ms) OF THE ALGORITHMS TO CONVERT FROM RASTER TO LHQT AND TLHQT. SPACE CORRESPONDS TO THE MAXIMUM SPACE TAKEN IN TERMS OF THE LENGTHS OF THE LISTS. S1 IS FOR THE MAXIMUM SPACE TAKEN BY THE ALGORITHM TO BUILD LHQT, WHILE S2 IS FOR THE EXTRA SPACE TAKEN BY THE CAND LISTS. $B$ REPRESENTS THE NUMBER OF BLACK NODES. THE IMAGES 2–7 HAVE THEIR LEVEL-WISE DISTRIBUTION OF QNODE'S SKEWED TOWARDS PIXELS.

| | Timing (ms) | | | Space | |
|---|---|---|---|---|---|
| S1 No. | Raster-LHQT | Raster-TLHQT | $B$ | S1 | S2 |
| 1 | 702 | 765 | 51 | 70 | 8 |
| 2 | 901 | 1147 | 418 | 424 | 31 |
| 3 | 913 | 1099 | 343 | 356 | 26 |
| 4 | 926 | 1076 | 256 | 292 | 18 |
| 5 | 986 | 1310 | 624 | 609 | 38 |
| 6 | 942 | 1250 | 585 | 582 | 34 |
| 7 | 811 | 1206 | 470 | 465 | 25 |

## IV. COMPUTATION OF GEOMETRIC PROPERTIES

We now use the TLHQT to compute three typical geometric properties (of binary images) that require exploration of adjacencies. They are: 1) connected component labeling, 2) perimeter, and 3) Euler number. We demonstrate that the adjacencies are easily explored using the threads, thus simplifying the respective algorithms.

### A. Connected Component Labeling

This is the process of assigning a label to all those QNODE's which form a connected set. The algorithms given in [6] and [7] accomplish the labeling in two steps:

1) The QNODE's are assigned initial labels with the constraint that the label assigned to a QNODE is propagated to all its neighbors. If a QNODE and its neighbor assume different labels, the two labels are put into an equivalence set.

2) The equivalence sets are merged to form equivalence classes [20], [21]. A unique label is assigned to all those QNODE's whose label is in a single equivalence class.

The proposed algorithm uses the TLHQT as the input and executes the two steps in a way different from and faster than the ones found in [6] and [7]. In fact, for a given QNODE, the adjacency in all four directions is easily explored using the threads. Once a link is established from a given QNODE Q1 to its neighbor Q2 in a given direction using a procedure EXPLORE_ADJ, both Q1 and Q2 are labeled. A label originating at the node Q1 is spread to the maximal area of the connected region by recursively invoking the procedure EXPLORE_ADJ for Q2. As a result, the number of equivalence sets of labels is minimized. A QNODE Q1 which is examined in all four directions has its label $> 0$ if Q1 is not isolated. This fact is made use of in avoiding any possible reconsideration of Q1 at a latter stage. Thus it can be seen that for the phase 1 of the labeling algorithm, each QNODE is visited only once. See Fig. 3 for an illustration of the procedure.

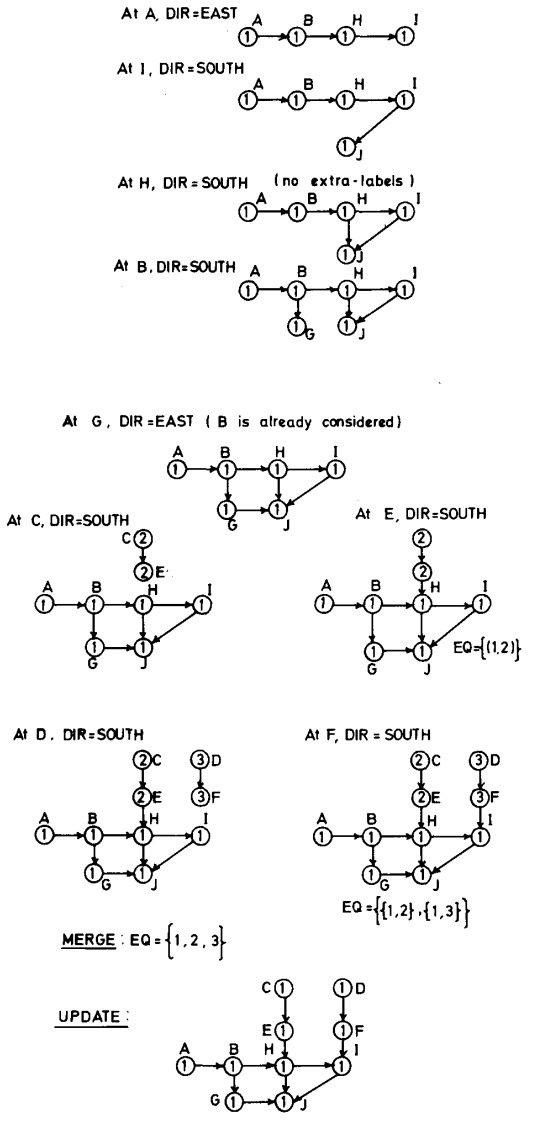



Fig. 3. An illustration of execution of the labeling algorithm.

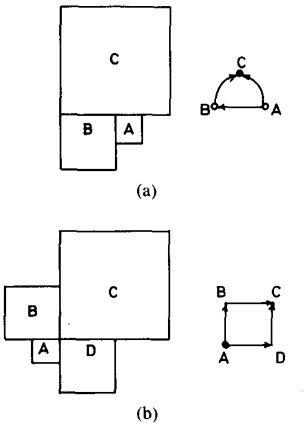



Fig. 4. An illustration of (a) TRIPLES and (b) QUADRUPLES.

### B. Perimeter Computation

Perimeter is the length of the boundary of a binary picture. While using the TLHQT for the computation of perimeter, we note that a QNODE at level $k$ can contribute to the perimeter an amount equal, at most, to $4 \times 2$. Whenever a QNODE has a link to a neighbor, twice the length of the segment shared by the QNODE and the neighbor is subtracted from the global variable PERIMETER. After the four directions have been examined, PERIMETER is updated by $4 \times 2$. Note that each QNODE is examined only once.

### C. Euler Number Computation

Euler number of an image is the difference between the number of connected components and the number of holes in the image. Dyer [9] has shown that the Euler number is given by $B - E + V$, where $B$ is the number of black nodes, $E$ the number of pairs of adjacent blocks, and $V$ the number of instances of three or four blocks sharing a corner without a white block touching the same corner. (We refer to these instances as triples and quadruples respectively. See Fig. 4.)

In a TLHQT, $B$ is the number of QNODE's and $E$ is the number of threads. Thus $V$ alone needs to be computed. To this end, consider Fig. 4(a) corresponding to the case of a triple. Each triple can be represented by a graph having three vertices and three links. A vertex is associated with a black block and an edge with a thread. Eight graphs are possible for triples, of which only four represent valid geometrical placements of blocks. All the four graphs and the corresponding placement of blocks are given in Fig. 5. It can be seen that there is a starting vertex in each graph, from which it is possible to reach a common vertex, moving first clockwise and then anticlockwise. Because of symmetry with respect to the starting vertex, only the graphs shown in Fig. 5(a) and (b) are considered for explaining the algorithm to detect triples.

The quadruple shown in Fig. 4(b) can also be represented similarly by a graph having four vertices and four links. Here only eight graphs represent valid placement of nodes and because of symmetry with respect to the starting vertex, only four cases shown in Fig. 6 need to be considered.

The basic idea behind the proposed algorithm using the TLHQT is to compute the number of instances of triples and quadruples in the image traversing the links originating from a QNODE. From Figs. 5 and 6, it follows that to locate a triple (or a quadruple) around the NORTH and EAST directions of a QNODE $Q$, one has to traverse the links in both directions. Starting along the NORTH direction, if one collects successive clockwise links in a set, the traversal terminates in at most three steps. Now if one starts from $Q$ and proceeds along the EAST direction, collecting successive anticlockwise links, one is bound to find a link that belongs to the first set, in cases shown in Fig. 5 and Fig. 6(a), (b), and (c). However, the case corresponding to the graph in Fig. 6(d) needs a slightly different treatment, since starting from the vertices $A$ and $C$, one cannot go beyond one step in the two perpendicular directions. Since $A$ and $C$ are considered one after the other, the decision regarding quadruple has to be deferred until both $A$ and $C$ are considered. The QNODE (say $A$) which is examined first will mark the two QNODE's $B$ and $D$ reachable from $A$; the other QNODE (say $C$) examined next will find $B$ and $D$ marked, detecting a quadruple. See Figs. 7 and 8 for typical outputs.

### D. Complexity of the Algorithms

It is easy to see that for the perimeter computation and for the first phase of the connected component labeling, each black node (i.e., a QNODE) is examined only once. For these two cases, the

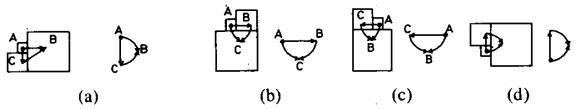



Fig. 5. Possible instances of TRIPLES.

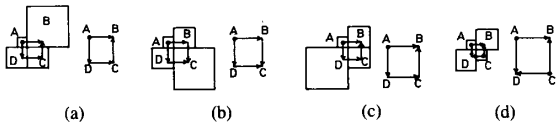



Fig. 6. Four of the eight possible instances of QUADRUPLES.

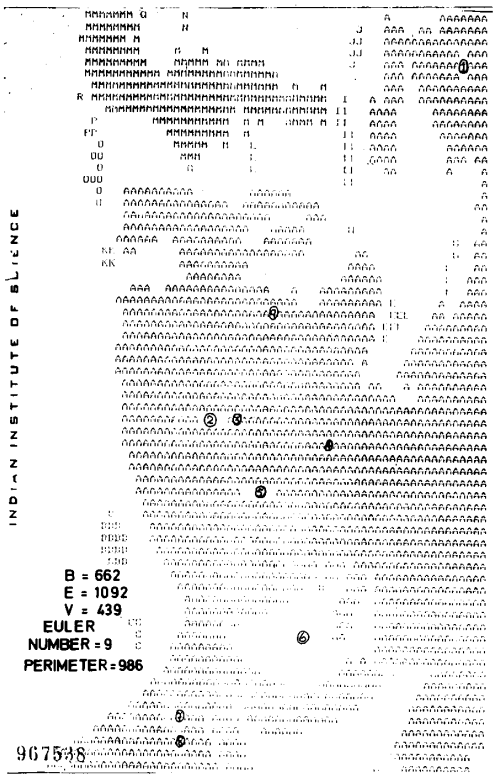



Fig. 7. Typical output for the algorithms of Section IV.

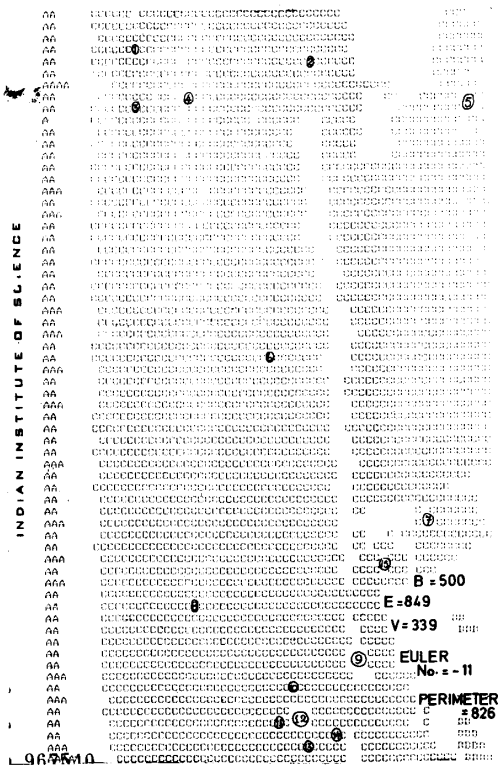



Fig. 8. Typical output for the algorithms of Section IV.

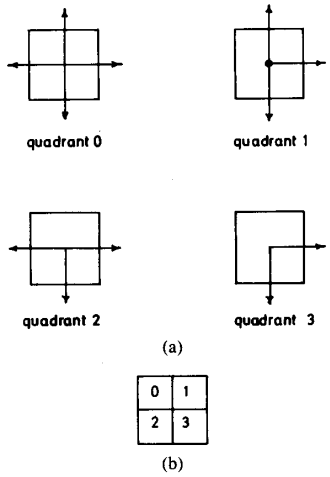



Fig. 9. (a) Arrangement of the maximum possible number of threads in each quadrant. (b) Arrangements of quadrants.

complexity is therefore $O(B)$. (However, as regards the second phase of the algorithm for the labeling, the time complexity is dependent on the merging technique and since it is a common factor for implementation of the same algorithm on other structures too, we have not considered it separately.) Also in the algorithm to compute the Euler number, around a QNODE, at the most only three instances of a triple or a quadruple are encountered; each such instance requires the traversal of at most four links. The algorithm also therefore has a time complexity of $O(B)$.

## V. Space Requirement of the TLHQT

The space consumed by the TLHQT is dependent on the total number of threads. Referring to Fig. 9, it can be seen that a given QNODE can occupy only one of the four quadrants indicated. The maximum number of links possible for each of these quadrants is indicated in Fig. 9. Therefore, on the average, the maximum number of threads possible out of a QNODE is only 3. Assuming that there are $B$ black QNODE's in the image and an extra word is necessary to store the LHQC of the QNODE, the average of the maximum space required will be (proportional to) $4B$. This may be contrasted with the storage requirement of the pointer-based quadtrees which is (proportional to) $8(B + W)$, assuming that each node requires 6 fields and that of the B-trees which is (proportional to) $8B$ [17].

TABLE IV
COMPARISON OF EXECUTION TIMINGS (ms) WITH THE ALGORITHMS OF SAMET AND TAMINNEN [16] ON THREE GEOMETRIC PROPERTIES COMPUTED. (IMAGES 2-7 HAVE THEIR BLACK NODES DISTRIBUTION SKEWED TOWARD PIXELS).

| | Perimeter | | | Euler Number | | Labeling | |
|---|---|---|---|---|---|---|---|
| Sl No. | B | Thread | ST | Thread | ST | Thread | ST |
| 1 | 56 | 5 | 21 | 7 | 33 | 12 | 43 |
| 2 | 418 | 33 | 109 | 90 | 151 | 138 | 172 |
| 3 | 343 | 27 | 87 | 79 | 119 | 102 | 153 |
| 4 | 256 | 21 | 55 | 62 | 76 | 73 | 117 |
| 5 | 624 | 48 | 173 | 131 | 238 | 230 | 308 |
| 6 | 585 | 46 | 143 | 134 | 198 | 208 | 286 |
| 7 | 470 | 37 | 130 | 112 | 149 | 154 | 201 |

*Legend*—Thread: TLHQT; ST: Samet and Taminnen algorithm.

TABLE V
COMPARISON OF EXECUTION TIMINGS (IN ms) OF THE LABELING ALGORITHM ON VARIOUS REPRESENTATION SCHEMES

| Sl No. | B | TLHQT (bottom-up) | LHQT | ST | LQ | PBQT |
|---|---|---|---|---|---|---|
| 1 | 56 | 12 | 73 | 43 | 570 | 38 |
| 2 | 418 | 138 | 439 | 172 | 1589 | 369 |
| 3 | 343 | 102 | 308 | 153 | 789 | 254 |
| 4 | 256 | 73 | 135 | 117 | 471 | 113 |
| 5 | 624 | 230 | 717 | 308 | 1572 | 588 |
| 6 | 585 | 208 | 659 | 286 | 1277 | 572 |

*Legend*—LQ: linear quadtree; ST: Samet and Taminnen [16]; PBQT: pointer-based quadtree.

## VI. CONCLUDING REMARKS

A variant of the linear quadtree, the threaded linear hierarchical quadtree (TLHQT) is proposed for representing binary images. The structure is shown to consume space of the $O(4B)$, where $B$ is the number of black nodes of the quadtree. Also by using this structure, efficient algorithms are presented for the computation of geometric properties of binary images. All the algorithms have a space requirement of $O(B)$; and the algorithms to compute perimeter and Euler number run in time $O(B)$, while the first phase of the labeling algorithm also executes in time $O(B)$. The performance of the algorithms to compute geometric properties is compared with other similar algorithms appearing in literature, in Tables IV and V. It is observed that the algorithms presented here perform better than other similar algorithms. It is also shown that the TLHQT can be built directly from a raster scan of the image, requiring an additional space and time (over what is necessary to build the LHQT) of $O(Bn^2/2^n)$ and $O(B)$, respectively.

## APPENDIX

We compute here the space and time for raster to TLHQT conversion over the raster to LHQT conversion, as this excess amount is what is required for the introduction of threads during raster scan. To this end, we derive three results here:

1) An estimate of the expected size of the list CAND at any level $k$, $0 \le k \le n-1$.

2) Maximum amount of space required to store the various CAND lists.

3) The expected amount of time required to introduce the threads.

From the algorithm, it is clear that all horizontal threads are introduced as and when a QNODE is built and hence no significant amount of space and time is required to introduce horizontal threads, when compared to the other phases of the algorithm. However, vertical threads at any level $k$, $0 \le k \le n-1$, of recursion are introduced after scanning two lists, CAND_TOP and CAND_BOT, respectively, for the top and bottom halves of the image segment, consisting of $2^k$ lines, corresponding to level $k$. Each element of each of these lists is scanned at most once and therefore the space and time required to introduce threads is directly related to the size of the list CAND at various stages of recursion.

We first state our assumptions:

1) The black nodes of the image are uniformly distributed over the image grid. Thus if $b_k$ is the expected number of black nodes at any level $k$, $0 \le k \le n-1$, corresponding to a level of recursion $k$ and in an area covered by $2^k$ lines of the raster is $b_k/2^{n-k}$.

2) In practice, there are many images which have their distribution of nodes skewed towards pixels. To derive a realistic complexity estimate, such a distribution of nodes expressible as shown below is assumed. Let $B$ be the total number of QNODE's of the TLHQT

$$b_k = Bp^{k+1}, \quad 0 < p < 1 \tag{1}$$

$$\text{s.t.} \quad p^* \sum_{k=1}^{n-1} p^k = 1. \tag{2}$$

It can be seen that the value of $p$ that satisfies (2) is nearly $0.5 + 1/3 * 2^n$. Accordingly, in the results derived below, the fact that $p \simeq 0.5 + 1/3 * 2^n$ is used in simplifying the expressions.

*Result 1:* The expected size of the list CAND at any level of recursion $k$, $1 \le k \le n-1$ is given by

$$C_k = \frac{Bp((2p)^{k+1} - 1)}{2^n(2p-1)}.$$

*Proof:* The list CAND at level $k$ will contain appropriate border node of sizes up to $4^k$. Therefore,

$$C_k = \sum_{i=0}^{k} b_i/2^{n-i}.$$

By Assumption 2, this reduces to

$$C_k = \sum_{i=0}^{k} \frac{Bp^{i+1}}{2^{n-i}} = \frac{Bp((2p)^{k+1-1})}{2^n(2p-1)}. \qquad \square$$

*Result 2:* The maximum space required to store various CAND lists is $O(n^2B/2^n)$.

*Proof:* It may be noted that at level $k$ of recursion, the QNODE's of level $k$ are inserted into the list CAND only after condensation at level $k+1$. So while estimating the space requirement, it is sufficient to consider $b_i$, $0 \le i \le k-1$ in deriving the value of $C_k$. Therefore from Result 1, it follows that

$$C_k = \frac{Bp((2p)^k - 1)}{2^n(2p-1)}.$$

The maximum number of CAND lists that can exist simultaneously, are present after the last two lines of image are analyzed. Fig. 10 illustrates this fact for a $16 \times 16$ image where the dark circles indicate the levels at which the CAND lists are active, i.e., remaining to be scanned. The expected maximum storage space for these lists is given by

$$\sum_{k=1}^{n-1} C_k + C_1 = \frac{Bp}{2^n(2p-1)} \sum_{k=1}^{n-1} ((2p)^k - 1) + \frac{Bp}{2^n},$$

using Result 1.

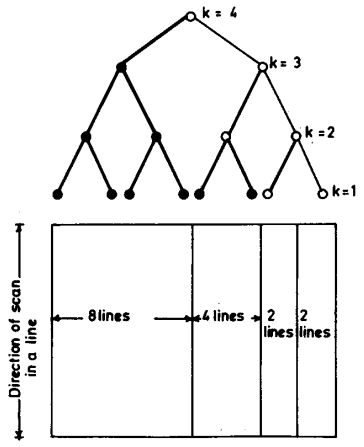



Fig. 10. A typical stage in the execution of the algorithm RASQUAD after scanning all the lines.

For $p \simeq 0.5 + 1/3 * 2^n$, the above sum reduces to

$$(3/2)\,Bp \sum_{k=1}^{n-1} (2p)^k - 1 + C_1$$
$$= (3/2)\,Bp \sum_{k=1}^{n-1} \left(1 + \frac{2}{3 * 2^n}\right)^k - 1\Bigg) + \frac{Bp}{2^n}$$
$$= \frac{3Bp}{2 * 2^n} \sum_{k=1}^{n-1} \frac{2k}{3 * 2^n} + \frac{Bp}{2^n},$$

for practical values of $n$ considered.

So,

$$\sum_{k=1}^{n-1} C_k + C_1 = \frac{B}{2^{n+1}} \sum_{k=1}^{n-1} k + \frac{Bp}{2^n}$$
$$= O\left(\frac{Bn^2}{2^n}\right). \qquad \square$$

*Result 3:* The amount of time required to introduce the vertical threads is $O(B)$.

*Proof:* At any level $k$, $2 \le k \le n$, of recursion, at any instance, two lists CAND_TOP and CAND_TOP produced from level $k-1$ are scanned. There are $2^{n-k}$ such instances corresponding to each level of recursion. So the time $T$ required to introduce vertical threads is proportional to $2^{n-k} \times 2 \times C_{k-1}$.

Due to the introduction of threads, the border QNODE's of level $k-1$, that do not condense, are in CAND_TOP or CAND_BOT. So the value of $C_k$ as given by Result 1 is used here.

So,

$$T = \sum_{k=2}^{n} 2^{n-k} 2 * C_{k-1}$$
$$= \sum_{k=2}^{n} 2^{n-k} * 2\,\frac{Bp((2p)^k - 1)}{2^n(2p-1)}, \qquad \text{using Result 1}$$
$$= \sum_{k=2}^{n} \frac{2Bp * 2^{n-k}}{2^n(2p-1)} \left(1 + \frac{2k}{3 * 2^n} - 1\right),$$
$$\text{using } p \simeq 0.5 + \frac{1}{3 * 2^n}$$

and considering practial values of $n$.

Therefore,

$$T = \frac{2Bp}{2^n} \sum_{k=2}^{n} k * 2^{n-k}$$
$$= \frac{2Bp}{2^n} (2^{n+1} - 4 - 2n) = O(B). \qquad \square$$

## Acknowledgment

The authors thank the anonymous referees whose suggestions greatly improved the presentation of the paper.

## References

[1] A. Klinger and C. R. Dyer, "Experiments in picture representation using regular decomposition," *Comput. Graphics Image Processing*, vol. 5, no. 1, pp. 68-105, Mar. 1976.
[2] H. Samet, "The quadtrees and related hierarchical data structures," *ACM Comput. Surveys*, vol. 16, no. 2, pp. 187-260, June 1984.
[3] I. Gargantini, "An effective way to represent quadtrees," *Commun. ACM*, vol. 25, no. 12, pp. 905-910, Dec. 1983.
[4] H. Samet, "Neighbouring finding techniques for images represented by quadtrees," *Comput. Graphics Image Processing*, vol. 18, no. 1, pp. 37-57, Jan. 1982.
[5] G. M. Hunter and K. Steiglitz, "Operations on images using quadtrees," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-1, no. 2, pp. 145-153, Apr. 1979.
[6] A. Unnikrishnan, Y. V. Venkatesh, and P. Shankar, "Connected component labelling using quadtrees—A bottom-up approach," *Comput. J.*, vol. 30, no. 2, pp. 176-182, Feb. 1987.
[7] H. Samet, "Connected component labelling using quadtrees," *J. ACM*, vol. 28, no. 3, pp. 487-501, July 1981.
[8] ——, "Computing perimeters of regions in images represented by quadtrees," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 3, no. 6, pp. 683-687, Nov. 1981.
[9] C. R. Dyer, "Computing the Euler number of an image from its quadtree," Comput. Graphics Image Processing, vol. 13, no. 3, pp. 270-276, July 1981.
[10] A. Unnikrishnan, Y. V. Venkatesh, and P. Shankar, "Distribution of black nodes at various levels in a linear quadtree," *Pattern Recognition Lett.*, vol. 6, pp. 341-342, Dec. 1987.
[11] A. Unnikrishnan and Y. V. Venkatesh, "An algorithm to convert a raster scanned image to linear hierarchical quadtrees," Dep. Elec. Eng., Indian Inst. Sci., Bangalore, India, Tech. Rep., May 1985.
[12] A. Unnikrishnan, P. Shankar, and Y. V. Venkatesh, "An algorithm to convert from rasters to threaded linear hierarchical quadtrees," Dep. Elec. Eng., Indian Inst. Sci., Bangalore, India, Tech. Rep., July 1985.
[13] L. Jones and S.S. Iyengar, "Space and time efficient virtual quadtrees," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-6, no. 2, pp. 244-247, Mar. 1984.
[14] E. Kawaguchi and T. Endo, "On a method of binary picture representation and its application to data compression," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-1, no. 1, pp. 27-35, Jan. 1980.
[15] M. A. Oliver and N. E. Wiseman, "Operations on quadtree encoded images," *Comput. J.*, vol. 26, no. 1, pp. 83-92, 1983.
[16] H. Samet and M. Tamminen, "Computing geometric properties of images represented by linear quadtrees," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-7, no. 2, pp. 229-239, Mar. 1985.
[17] D. J. Abel, "A B-tree structure for large quadtrees," Comput. Vision, Graphics, Image Processing, vol. 27, no. 1, pp. 19-31, July 1984.
[18] H. Samet, "An algorithm for converting rasters to quadtrees," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-3, no. 1, pp. 93-95, Jan. 1981.
[19] C. A. Shaffer and H. Samet, "Optimal quadtree construction algorithms," Comput. Vision, Graphics, Image Processing, pp. 402-419, Mar. 1987.
[20] D. E. Knuth, *The Art of Computer Programming, Vol. 1, Fundamental Algorithms*, 2nd ed. Reading, MA: Addison-Wesley, 1975.
[21] R. E. Tarjan, "Efficiency of a good but not linear set union algorithm," *J. ACM*, vol. 22, no. 2, pp. 215-225, Apr. 1975.