A NEW APPROACH TO LINEAR QUADTREES

WAYNE A. DAVIS and XIAONING WANG
Department of Computing Science
The University of Alberta
Edmonton, Alberta T6G 2H1

ABSTRACT[1]

This paper introduces a modified version of the linear quadtree [8] for region representations. The modified linear quadtree (**MLQ**) uses only (2n+log n)/3n of the memory space required by the linear quadtree to represent the same region, where n is the resolution parameter of the image. The following results will be presented. An **MLQ** can be constructed, given the array description of the image, in time proportional to the number of pixels in the image. Computing a neighbor node and determining its color can be done in time $O(n)$ and $O(\log N)$ respectively, where N is the number of nodes in the **MLQ**. Finally, an $O(n \cdot N)$ algorithm is given to generate the background of the region.

1. INTRODUCTION

Region representation is an important issue in image processing, cartography, computer graphics, geographic information system, and robotics. There are a variety of approaches in use. The success of such an approach to region representations depends on how easy to implement it in terms of computational complexity, memory space requirements, and extensibility.

Recently, quadtrees have received increasing attention. The quadtree region representation is based on the principle of recursive decomposition of an image proposed by Klinger [2]. It is relatively compact, and well-suited to operations such as complement, union and intersection [3], as well as the detection of various region properties [4,5,6,7].

Efforts to further reduce the space requirements of quadtrees have led to the concept of linear quadtrees [8]. In linear quadtrees, the use of a locational code or key provides a unique identification of the BLACK node associated with that key, and the quadtree topology can be obtained from the keys. As a result, a linear quadtree can represent a quadtree as a sequence of terminal nodes in a specific key order, while nonterminal nodes, or even WHITE nodes of the quadtree are omitted, achieving space efficiency.

In this direction, linear quadtrees proposed by Mark and Lauzon [10] stores both BLACK and WHITE nodes in a compressed form that only the key of the last terminal node in a sequence of consecutive terminal nodes of the same color is stored. The linear quadtree of Gargantini [8] stores only BLACK nodes comprising the region, and generates WHITE nodes when required, where each BLACK node is associated with an n digit quaternary code whose digits reflect the successive decomposition of a 2^n by 2^n image. As reported in [8], such a representation introduces a saving of at least 66 percent of the memory space required by a quadtree.

The purpose of this paper is to present a modified version of Gargantini's linear quadtree to achieve further efficiency in terms of storage requirements and execution time. The recursive decomposition of images is briefly reviewed in Section 2 along with quadtree and linear quadtree representations in preparation for the development and discussion of an alternative representation. The modified linear quadtree (**MLQ**) is introduced in Section 3, and is shown to be more efficient than a linear quadtree spacewise. Section 4 investigates some basic operations on images using the proposed structure.

---

[1] This research was supported in part by the Natural Sciences and Engineering Research Council of Canada under Grant NSERC A7634.

2. BACKGROUND

Definition 1: A binary **image** is a 2^n by 2^n array of unit square pixels each of which can assume either BLACK or WHITE values, where n is called the **resolution parameter** of the image.

Definition 2: The **region** of an binary image is composed of all BLACK pixels, and the **background** of the region is composed of all WHITE pixels.

Example: The region shown in Fig. 1 is represented by the 2^3 by 2^3 binary array in Fig. 2, where 1 and 0 correspond to BLACK and WHITE pixels, respectively.

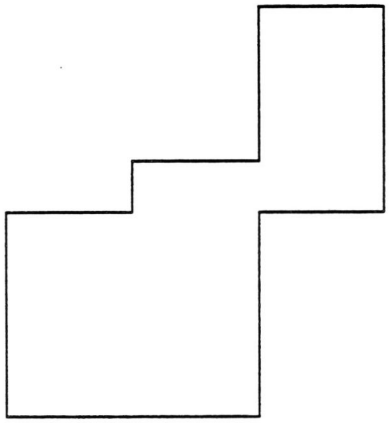

Fig. 1. A region.

7	0	0	0	0	1	1	0	0
6	0	0	0	0	1	1	0	0
5	0	0	0	0	1	1	0	0
4	0	0	1	1	1	1	0	0
3	1	1	1	1	0	0	0	0
2	1	1	1	1	0	0	0	0
1	1	1	1	1	0	0	0	0
0	1	1	1	1	0	0	0	0
	0	1	2	3	4	5	6	7

Fig. 2. The region in binary array representation.

Definition 3: Let (i,j) be a pixel of a given image. Then (i,j) has four horizontal and vertical neighbors: (i-1,j), (i,j-1), (i,j+1) and (i+1,j). These pixels are called the **neighbors** of (i,j).

There has been considerable interest in region representations based on the principle of a recursive decomposition. An image is decomposed in the following manner: if the region does not cover the entire binary array, the array will be subdivided into four equal-sized square blocks, this process will be applied recursively to each block, till blocks are obtained that are totally contained in the region or totally disjoint from it. As an example, Fig. 3 is the decomposition of the region shown in Fig. 1.

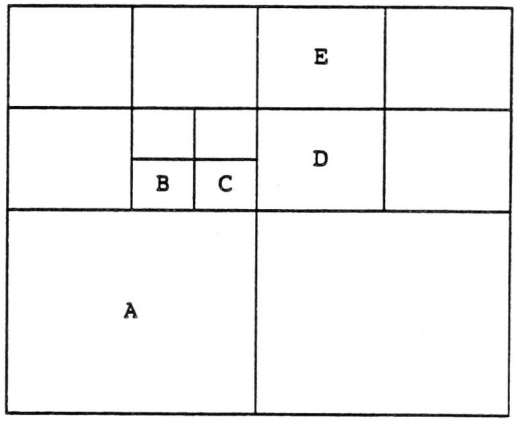

Fig. 3. Block decomposition of the region in Fig. 1.

The recursive decomposition of the image results in blocks that must have standard sizes and positions. Similar definitions can be formulated in terms of blocks. For example, a block is said BLACK if it contains only BLACK pixels, WHITE if it contains only WHITE pixels, and GREY if it contains both BLACK and WHITE pixels.

A quadtree is an ordered tree of degree four. The root represents the entire image, and each other node represents one of the four subblocks in order NW, NE, SW, SE of its father's block. No father node can have all its descendant terminal nodes with the same

color. Fig. 4 demonstrates the quadtree representation for the region in Fig. 1, where the symbols O, □ and ■ represent GREY, WHITE and BLACK nodes, respectively. The terms block and node will be used interchangeably throughout.

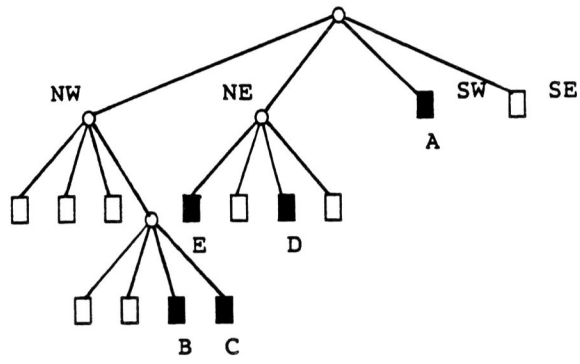

Fig. 4. The quadtree representation for the region in Fig 1.

Each node in the quadtree has five pointers: one to its father and four to its sons. Terminal nodes have color BLACK or WHITE while nonterminal nodes have color GREY.

The distinct feature of the linear quadtree is pointerless and stores only BLACK nodes. To encode BLACK nodes, the NW quadrant is encoded with 0, the NE with 1, the SW with 2, and the SE with 3. Each BLACK pixel is then encoded by a sequence of n integers (i.e., quaternary code): $q_{n-1}, q_{n-2}, \cdots, q_0$, where $q_i \epsilon \{0,1,2,3\}$ for $0 \le i < n$.

Each successive integer represents the block decomposition from which it originates. Thus, q_{n-i}, $1 \le i \le n$, identifies the subblock to which the pixel belongs at the ith decomposition. A BLACK node corresponding to a 2^k by 2^k block is encoded by $q_{n-1}, q_{n-2}, \cdots, q_0$, with $q_i = X$, $X \neq 0,1,2,3$, for $0 \le i < k$. X is called a don't care sign. A linear quadtree is an array of BLACK nodes comprising the region sorted in the quaternary code order. As an example, the liner quadtree representation for the region in Fig. 1 is given by the following sequence:

032, 033, 10X, 12X, 2XX

which correspond to the BLACK nodes B, C, E, D, and A in Fig 3.

A number of algorithms supporting the linear quadtree have been reported in [8,9], and they suffer, in speed, from the above quaternary code encoding of the BLACK nodes.

3. THE MODIFIED LINEAR QUADTREE

The following conventions will be adopted throughout.

Definition 4: For integers I and J given by

$$I = \sum_{i=0}^{n-1} (I_i \cdot 2^i) \text{ and } J = \sum_{i=0}^{n-1} (J_i \cdot 2^i),$$

where $I_i, J_i \epsilon \{0,1\}$,

$$\mathbf{SHUFFLE}(I,J) = \sum_{i=0}^{n-1} (I_i \cdot 2 + J_i) \cdot 4^i.$$

Definition 5: For an integer $S = s_{2n-1} \cdots s_0$, where $s_i \epsilon \{0,1\}$,

$$\mathbf{EVEN}(S) = \sum_{i=0}^{n-1} s_{2i} \cdot 2^i, \text{ and}$$

$$\mathbf{ODD}(S) = \sum_{i=0}^{n-1} s_{2i+1} \cdot 2^i.$$

Hereafter, **SH**, **EV** and **OD** will be used as abbreviated versions of **SHUFFLE**, **EVEN** and **ODD**, respectively.

There are a number of ways to assign consecutive integers to the pixels of a 2^n by 2^n image. A method which is now known as a Morton matrix [1], is the best to capture the nature of the recursive decomposition process. A Morton matrix is an array of 2^n by 2^n pixels each of which is assigned an integer as follows. For a pixel p with coordinates (I,J), where I and J are the column and row position, respectively, the integer assigned to p is given by **SH**(I,J).

The integer assigned to a pixel is termed the key of the pixel. To represent a block obtained by the recursive decomposition method discussed in Section 2 requires:

Definition 6: The **key** of a block or node is the key of its left lower pixel, and the resolution parameter of the block is an integer s such that the size of the block is 2^s.

Example: Fig. 5 shows a 2^3 by 2^3 Morton matrix.

7	21	23	29	31	53	55	61	63
6	20	22	28	30	52	54	60	62
5	17	19	25	27	49	51	57	59
4	16	18	24	26	48	50	56	58
3	5	7	13	15	37	39	45	47
2	4	6	12	14	36	38	44	46
1	1	3	9	11	33	35	41	43
0	0	2	8	10	32	34	40	42
	0	1	2	3	4	5	6	7

Fig. 5. A 2^3 by 2^3 Morton matrix.

The two-tuple <K,s> will uniquely represent a block, where K and s respectively are the key and the resolution parameter of the block. For a given block Q, its quadrant with label i is called the ith quadrant of Q and denoted as Q_i, where $i \in \{0,1,2,3\}$. The quadrant labeling shown in Fig. 6 will be assumed.

1	3
0	2

Fig. 6. Quadrant Labeling.

The following two lemmas are useful in developing successive algorithms.

Lemma 1: For any two nodes $<K_1,s_1>$ and $<K_2,s_2>$ where $<K_1,s_1> \in Q_i$ and $<K_2,s_2> \in Q_j$ for some node Q, if i<j then $K_1<K_2$.

Lemma 2: For any two nodes $<K_1,s_1>$ and $<K_2,s_2>$ where $s_1>s_2$, then either $<K_1,s_1>$ contains $<K_2,s_2>$, or the intersection of $<K_1,s_1>$ and $<K_2,s_2>$ is empty.

3.1. An encoding scheme

A BLACK node can now be encoded by a two-tuple <K,s>, where K uniquely identifies the position of the node in the image, and s specifies the size of the node. A modified linear quadtree is defined as a sorted sequence of BLACK nodes in their two-tuple form in ascending key order. This differs from the linear quadtree in that, firstly, the key of the node is stored as a single integer rather than an n digit quaternary code, and, secondly, the resolution parameter of the node is explicitly expressed rather than implied by the number of don't cares in the quaternary code.

Such a modification results in two advantages: space efficiency and improved execution time. The second advantage will become clear in [11]. To see the first advantage necessitates the comparison between the storage requirements of the two encoding methods. As reported in [8], each BLACK node needs 3n bits. By contrast, the proposed encoding scheme requires (2n+log n) bits for each BLACK node, where 2n bits are used for storing the key and (log n) bits for the resolution parameter of the node. As an example, when n=12, the **MLQ** can save approximately 22 percent of the memory space required by the linear quadtree.

3.2. Constructing an MLQ

A recursive algorithm to be introduced in this section constructs an **MLQ**, given the array description of an image. An optimal algorithm will be presented in [11] for obtaining an **MLQ**, given the quadtree description of the image. Such an algorithm is useful because obtaining an **MLQ** from a quadtree description of an image costs much less both in time and space than generating an **MLQ** from an array description of the same image.

The algorithm for generating an **MLQ** from the array description of a 2^n by 2^n binary image is presented below as a procedure termed **ARR-TO-MLQ**. It takes three input parameters E, key and s, where E is a 2^s by 2^s binary array, key and s initially correspond to zero and n, respectively. The output of the algorithm is a globle variable called LIST containing the desired **MLQ**.

The algorithm examines each pixel element of the binary image in Morton sequence order. If a pixel is BLACK, then its two-tuple representation is formed and added to LIST. One of the key features is that the algorithm merges the four small BLACK nodes corresponding to the last four two-tuples in LIST to yield

a bigger BLACK node whenever it is possible by replacing the four two-tuples by one representing the bigger BLACK node. Upon termination of the algorithm, LIST contains all maximal BLACK nodes. This is in constrast with the algorithm that first builds a list containing all BLACK pixels, and then attempts to merge the pixels in the list into maximal BLACK nodes [8].

As an example, if the algorithm is applied to the binary image in Fig. 2, the **MLQ** representing the region in Fig. 1 is obtained:
<0,2>, <24,0>, <26,0>, <48,1>, <52,1>, which corresponds to the BLACK nodes A, B, C, D, E of Fig. 3, respectively.

```
Procedure ARR-TO-MLQ(E,key,s)
 begin
   if s=0 then
    begin
     if E is a BLACK pixel then
      begin
       add pair <key,s> to LIST;
       return(BLACK);
      end
     else return(NONBLACK)
    end
   else
    begin
     for i:=0 to 3 do
      color[i]:=
      ARR-TO-MLQ(E_i,key+i*4**(s-1),s-1);
     if color[i] is BLACK, 0≤i≤3, then
      begin
       replace the last 4 pairs
       in LIST by <key,s>;
       return(BLACK);
      end
     else return(NONBLACK);
    end;
 end;
```

Theorem 1: The above method constructs an **MLQ**, given the array description of the image, in time linear to the number of pixels in the image.
Proof: Let $T(4^n)$ be the number of steps required by procedure **ARR-TO-MLQ** to generate an **MLQ**. Clearly, $T(1)=1$. If $n>0$, $T(4^n)$ is the total number of steps used in the four calls of **ARR-TO-LMQ** on an array of size 4^{n-1}, plus approximately the four steps in checking through color[0] to color[3]. That is,

$$T(4^n) = \begin{cases} 4T(4^{n-1})+4 & n>0 \\ 1 & n=0. \end{cases}$$

The Theorem follows by solving the recurrence.

Q.E.D.

4. BASIC ALGORITHMS

4.1. Neighbor finding

Neighbor finding for regions represented by an **MLQ** is a fundamental operation. It is the cornerstone for many applications such as labeling connected components, computing perimeters, and others, which will be discussed in [11]. Neighbor finding using **MLQs** involves essentially two steps. The first step obtains the two-tuple representation of the desired neighboring block from that of a given block. The second step then determines the color of the neighboring block by consulting the **MLQ** under consideration. These two steps will now be described more precisely.

4.1.1. Neighbor determination

Let each node in an **MLQ** be stored as a record containing two fields. The first field, named KEY, contains the key of the node. The second field, named RES, contains the resolution parameter of the node. The use of **OD** and **EV** functions on the key of the node will provide the coordinate information of the node. This coordinate information is sufficient to obtain the coordinates of a neighboring block in a specified direction. Then the two-tuple form of the neighboring block can be constructed by using the **SH** function.

Let **P** be a given node, its neighbor of equal size in a direction specified by **side** can be determined by the following procedure termed **EQ-NEIGHBOR** using the 4-adjacent criterion. Note that determining a neighbor of different size or using other adjacency criterion can be done similarly.

```
Procedure EQ-NEIGHBOR(P,side)
  begin
    I:=OD(P.KEY);
    J:=EV(P.KEY);
    case of side
      'N': J:=J + 2**P.RES;
      'E': I:=I + 2**P.RES;
      'S': J:=J - 2**P.RES;
      'W': I:=I - 2**P.RES;
    end;
    neighbor.KEY:=SH(I,J);
    neighbor.RES:=P.RES;
    return(neighbor);
  end;
```

Theorem 2: The time complexity of constructing a neighboring node in any of the principal directions is $O(n)$.
Proof: The time complexities of the functions **SH**, **EV** and **OD** are all of $O(n)$.
Q.E.D.

4.1.2. Color determination

To determine the color of the neighboring node necessitates searching the **MLQ**, since only BLACK nodes are explicitly stored there. Let **X** be the neighboring node which is to be compared with a node **Y** of the **MLQ** in the course of the binary search. The color of **X** can then be found as one of the following cases according to Lemma 2:

1. If **X.KEY** and **X.RES** are identical to **Y.KEY** and **Y.RES**, respectively, then **X=Y** and **X** is BLACK.

2. If both of the following conditions are satisfied:
   a) **X.RES<Y.RES**,
   b) **Y.KEY≤X.KEY<Y.KEY+4**Y.RES**,
then **X** is contained in **Y**, and therefore **X** is BLACK.

3. If both of the following conditions are satisfied:
   a) **X.RES>Y.RES**,
   b) **X.KEY≤Y.KEY<X.KEY+4**X.RES**,
then **Y** is contained in **X**, and therefore **X** is GREY.

4. If none of the above cases holds, then it follows that **X** is WHITE.

Theorem 3: Determining the color of a given node can be done in $O(\log N)$ steps, where N is the number of BLACK nodes in the **MLQ**.
Proof: Due to the time complexity of the binary search.
Q.E.D.

4.2. Complement

This section describes an $O(n \cdot N)$ method to complement a region represented by an **MLQ**. The complement of the region of a binary image is the logical complement of the binary image. In other words, the complement of a region is the background of the region. In an **MLQ** region representation, such an operation is useful because WHITE nodes comprising the background are not explicitly stored. In what follows, an algorithm is given, which is capable of infering all maximal WHITE nodes in ascending key order from two given consecutive BLACK nodes. Therefore, no further sorting and condensing are necessary.

The algorthm consists of four parts corresponding to four procedures called **GREYNODE**, **WHITE1**, **WHITE2** and **WHITE3**. Let **E** represent the entire image, i.e., **E.KEY=0** and **E.RES=n**. Procedure **GREYNODE** takes, as input, two BLACK nodes **A**, **B** and the entire image **E** to locate a GREY node **Q** recursively such that $A \epsilon Q_i$, $B \epsilon Q_j$, for $i \neq j$, and returns the value of Q_i and Q_j to **C** and **D**, respectively, as output.

Procedure **WHITE1** takes, as input, a BLACK node **B** and a node **Q**, which is either GREY or BLACK, with $B \epsilon Q$, and recursively generates all maximal WHITE nodes within **Q** in ascending key order except those whose keys are less than that of **B**.

Procedure **WHITE2** takes, as input, two nodes Q_i and Q_j for some **Q**, and generates, as output, WHITE nodes Q_k for all $i<k<j$ in ascending key order.

Procedure **WHITE3** takes, as input, a BLACK node **B** and a node **Q**, which is either GREY or BLACK, with $B \epsilon Q$, and generates recursively all maximal WHITE nodes within **Q** in ascending key order except those whose keys are greater than that of **B**.

The algorithm is given as procedure **WHITENODES**. The input of the algorithm is two BLACK nodes b1 and b2 with b1.KEY < b2.KEY. The output of the algorithm is a list containing all maximal WHITE nodes generated by **WHITE1**, followed by those generated by **WHITE2**, followed by those generated by **WHITE3**. By lemma 1, the obtained sequence of WHITE nodes between b1 and b2 is in ascending key order.

```
Procedure GREYNODE(A,B,E,C,D)
  begin
    determine i and j
    such that AєEi and BєEj;
    if i=j then GREYNODE(A,B,Ei,C,D)
           else begin
                  C=Ei;
                  D=Ej;
                end;
  end;
```

```
Procedure WHITE1(B,Q)
  begin
    if B and Q are different
      then begin
             determine i such that B∈Qi;
             WHITE1(B,Qi);
             for k=i+1 to 3 do
               add Qk to list;
           end;
   end;

Procedure WHITE2(Qi,Qj)
  begin
    for k=i+1 to j-1 do
      add Qk to list;
  end;

Procedure WHITE3(B,Q)
  begin
    if B and Q are different
      then begin
             determine j such that B∈Qj;
             for k=0 to j-1 do
               add Qk to list;
             WHITE3(B,Qj);
           end;
  end;

Procedure WHITENODES(b1,b2)
  begin
    GREYNODE(b1,b2,E,Q1,Q2);
    WHITE1(b1,Q1);
    WHITE2(Q1,Q2);
    WHITE3(b2,Q2);
  end.
```

Theorem 4: The time complexity of Algorithm **WHITENODES** is $O(n)$.
Proof: The purpose of procedure **GREYNODE** is to locate a GREY node **Q** such that b1 and b2 belong to two different quadrants of **Q**. Such a **Q** in conjunction with b1 and b2 will be sufficient to infer all maximal WHITE nodes between b1 and b2. The order in which **WHITE1**, **WHITE2** and **WHITE3** appear in the algorithm is the consequence of lemma 1.

Let n_0 be the depth of recursion of **GREYNODE**, then

$$1 \le n_0 \le n - m, \quad \text{where } m=\max\{b1.RES, b2.RES\}. \qquad (1)$$

n_0 is bounded from below by 1, simply because the **Q** located can be just the entire image **E** itself, while n_0 is bounded from above by n - m due to the following reasons. When **GREYNODE** is to be recursively called for the (n - m)-th time, a node of size 2^{m+1} is being taken as the **Q** which is so far the smallest GREY node to which both b1 and b2 belong. However, the sizes of b1 and b2 require that they now must belong to two different quadrants of **Q**. Therefore no further recursion is necessary.

Let n_1 be the depth of recursion of **WHITE1**, then

$$n_1 = n - n_0 - s_1, \quad \text{where } s_1=b1.RES. \qquad (2)$$

In fact, the node **Q1** obtained from **GREYNODE** is of size $2^{**}(n-n_0)$, and when this **Q1** is recursively subdivided by **WHITE1** into a node of size $2^{**}s_1$, the recursion will terminate. This explains (2).

Similarly, let n_3 be the depth of recursion of **WHITE3**, then

$$n_3 = n - n_0 - s_2, \quad \text{where } s_2=b2.RES. \qquad (3)$$

The total cost **T** in terms of the depth of recursion required is, therefore, $n_0 + n_1 + 1 + n_3$, where 1 originates from **WHITE2**. Thus

$$T = 2n - n_0 - (s_1+s_2) + 1. \qquad (4)$$

Substituting (1) back into the right hand side of (4) yields:

$$n+m-(s_1+s_2)-1 \le T \le 2n-(s_1+s_2). \qquad (5)$$

Since the total number of recursion required is less than 2n, according to (5), and each recursion takes only constant time. Theorem 4 then follows.

Q.E.D.

With algorithm **WHITENODES**, it is now possible to generate all maximal WHITE nodes comprising the background of the region in ascending key order when traversing the **MLQ**.

The following procedure termed **COMPLEMENT** takes, as input, **M** and N corresponding to the **MLQ** and the number of BLACK nodes in **MLQ**, respectively. The output of the algorithm is a list containing the maximal WHITE nodes in ascending key order.

In procedure **COMPLEMENT**, **WHITE3(M[1],E)** enumerates all WHITE nodes in **E** whose keys are less than **M[1].KEY**, while **WHITE3(M[N],E)** enumerates all WHITE nodes in **E** whose keys are greater than **M[N].KEY**. The for loop generates WHITE nodes between every two consecutive BLACK nodes of the **MLQ**.

```
Procedure COMPLEMENT(M,N)
  begin
    WHITE3(M[1],E);
    for i:=1 to N-1 do
      WHITENODES(M[i],M[i+1]);
    WHITE1(M[N],E);
  end;
```

The following result follows directly from Theorem 4.
Theorem 5: The time complexity of **COMPLEMENT** is $O(n \cdot N)$.

5. CONCLUSIONS

A data structure called **MLQ** for region representations has been presented in this paper. Basic operations on images for an **MLQ** have been developed to support the proposed data structure. It has been shown that the **MLQ** approach requires less storage than the linear quadtree approach. The results in [11] will demonstrate that an **MLQ** is more efficient than a linear quadtree in terms of time complexities.

REFERENCES

1. Morton, G.M. "A Computer Oriented Geodetic Data Base, and a New Technique in File Sequencing", *IBM Canada Limited, unpublished report*, March 1, 1966.

2. Klinger, A. and Dyer, C.R. "Experiments in Picture Representation Using Regular Decomposition", *Comput. Graphics and Image Process.*, Vol. 5, pp. 68-105, 1976.

3. Hunter, G.M. and Steiglitz, K. "Operations on Images Using Quad Trees", *IEEE Trans. Pattern Analy. & Mach. Intell.*, Vol PAMI-1, pp. 145-153, 1979.

4. Dyer, C.R., Rosenfeld, A. and Samet, H. "Region Representation: Boundary Codes from Quadtrees", *Comm. ACM*, Vol. 23, pp. 171-179, March 1980.

5. Shneier, M. "Note: Calculations of Geometric Properties Using Quadtrees", *Comput. Graphics and Image Process.*, Vol. 16, pp. 296-302, 1981.

6. Samet, H. "Connected Component Labeling Using Quadtrees", *J.ACM*, Vol. 28, pp. 487-501, 1981.

7. Samet, H. "Neighbor Finding Techniques for Images Represented by Quadtrees", *Comput. Graphics and Image Process.*, Vol. 18, pp. 37-57, 1982.

8. Gargantini, I. "An efficient way to represent properties of quadtrees", *Comm. ACM*, Vol. 25, pp. 905-910, Dec. 1982.

9. Gargantini, I. "Translation, rotation and superposition of linear quadtrees", *Int. J. Man-Mach. Stud.*, Vol. 18, pp. 253-263, March 1983.

10. Mark, D.M. and Lauzon, J.P. "Linear Quadtrees for Geographic Information Systems", *Proc. of the International Symposium on Spatial Data Handling*, Vol. 2, pp. 412-430, Zurich, Switzerland, Aug. 20-24, 1984.

11. Wang, X. "Some New Approaches for Linear Quadtrees", M.Sc. Thesis, in preparation.