PERGAMON

Pattern Recognition 34 (2001) 631–639

PATTERN RECOGNITION
THE JOURNAL OF THE PATTERN RECOGNITION SOCIETY

www.elsevier.com/locate/patcog

An adaptive algorithm for conversion from quadtree to chain codes

Frank Y. Shih*, Wai-Tak Wong

Department of Computer and Information Science, Computer Vision Laboratory, New Jersey Institute of Technology, Newark, NJ 07102, USA

Received 17 August 1999; received in revised form 16 December 1999

Abstract

An adaptive algorithm is presented for converting the quadtree representation of a binary image to its chain code representation. Our algorithm has the advantage of constructing the chain codes of the resulting quadtree of the Boolean operation of two quadtrees by re-using the original chain codes. This algorithm is adaptive because it can adjust the total number of internal nodes to be stored and retrieve it later in the reconstruction stage. The algorithm possesses parallelism and is suited for pyramid architecture. Our algorithm requires time $O(H + L)$ in sequential and time $O(N)$ in parallel, where H is the height of the quadtree, L is the length of the chain code sequence generated, and $N \times N$ is the size of the input image. © 2001 Pattern Recognition Society. Published by Elsevier Science Ltd. All rights reserved.

Keywords: Image representation; Chain code; Quadtree; Parallel algorithm; Adaptive conversion

1. Introduction

Representation and manipulation of digital images are two important tasks in image processing, pattern recognition, pictorial database, computer graphics, geographic information systems, and other related applications. Quadtree is one of the compact hierarchical data structures in representing a binary image [1]. It is constructed by successively sub-dividing an image into four equal-size sub-images in the NW (northwest), NE (northeast), SW (southwest), and SE (southeast) quadrants. A homogeneously colored quadrant is represented by a leaf node in the tree. Otherwise, the quadrant is represented by an internal node and further divided into four subquadrants until each subquadrant has the same color. The leaf node with a black (white) color is called the black (white) node and the internal node is called the gray node. An example is shown in Fig. 1 where the alphabet labels are explained in Section 5. There are two widely-used representations of quadtrees. A pointer-based quadtree uses the standard tree representation. A linear quadtree can be either a preorder traversal of the nodes of a quadtree or the sorted sequence of the quadtree's leaves.

The quadtree's data structure has been applied to the representation of maps in the geographic information systems (GIS) successfully [2]. A fundamental operation is to overlay two maps with the union, intersection, or difference. Some research areas are the parallel computational models of the quadtree algorithm [3,4].

Another widely used method to represent digital images is contour representation. A binary image may contain multiple regions. The contour of each region is represented by a starting point and a sequence of moves around the borders. Freeman chain code [5] is one of the common coding techniques. It was developed not aiming at efficient image compression, but rather at making the processing and analysis simpler. The chain code moves along a sequence of border pixels based on 4- or 8-connectivity. The direction of each movement is encoded by using a numbering scheme, such as $\{i \mid i = 0, 1, \ldots, 7\}$ denoting an angle of $45i^\circ$ counter-clockwise from the positive x-axis. The chain codes can be viewed as a connected sequence of straight line segments with specified lengths and directions.

Basically, the chain coding algorithm is a sequential approach of contour tracing. It traces the border pixels one-by-one and generates codes by considering neighborhood allocation. A single-pass mid-crack coding

* Corresponding author. Tel.: + 1-973-596-5654; fax: + 1-973-596-5777.

E-mail address: shih@cis.njit.edu (F.Y. Shih).

0031-3203/01/$20.00 © 2001 Pattern Recognition Society. Published by Elsevier Science Ltd. All rights reserved.
PII: S0031-3203(00)00008-X

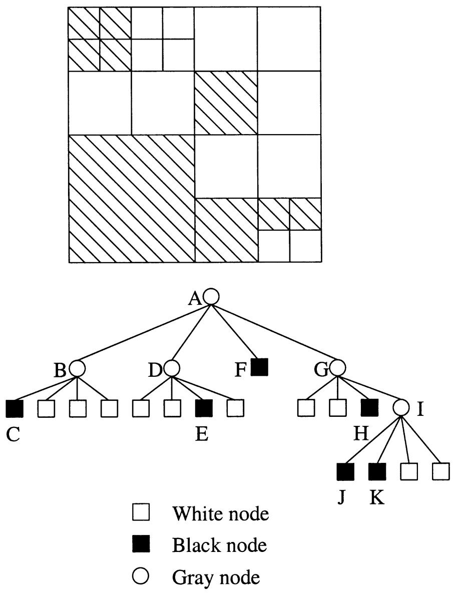


Fig. 1. An image and its quadtree representation.

algorithm [6] was developed to use a 3×3 template for the code extraction in a raster-scan fashion. It only requires a single row-by-row scan to generate all the code sequences even for a binary image composed of multiple regions. It has an advantage of detecting the spatial relationship between object regions. This algorithm can generate chain codes or mid-crack codes by adopting different look-up tables [7]. Based on the similar concept of extracting the chain codes first and then linking them together, Zingaretti et al. [8] presented another fast chain coding method.

Since both quadtree (region representation) and chain code (contour representation) have own advantages, it is of interests to develop a method of converting from one representation to another. Existing algorithms from chain code to quadtree conversion were found in Refs. [9,10]. The conversion from quadtree to chain code was reported by Dyer et al. [11] and Kumar et al. [12]. Both algorithms use the same approach as follows.

1. Find the starting node.
2. Extract the chain code of the current node.
3. Search the neighbors of the node in step 2.
4. Mark the edge between two nodes in steps 2 and 3. If the edge has been visited, the process is terminated. Otherwise, go back to step 2.

Our adaptive conversion algorithm from quadtree to chain code is based on the 8-neighbor traversal algorithm [13]. When the traversal algorithm reaches a node at the lowest level, we perform the chain coding process if the pixel is black. Chains are generated for each object pixel. When the traversal algorithm returns from the lowest level back to the root, we perform a merging process for chains that come from different nodes in each level. For example, an internal node will have four children and all chains that come from them are merged. The result of the merging process is a chain set. Then we return the resulting chain set to the parent of this internal node. Finally, we will have a chain set which represents the whole image when the root node is reached.

Unlike the existing conversion algorithms, our new algorithm has the superior capability of processing the multiple regions of an input image without additional effort. If we want to extract the chain code information from the set union, intersection, or difference of two quadtrees, our algorithm has the capability of reconstructing the new chain code information easily. This algorithm is suited for parallelization in pyramid architecture because it is based on the recursive function call from one node to its four subquadrants.

In the next section, we will present traversal algorithm for the pointer-based quadtree. Chain coding for the traversal algorithm is described in Section 3. In Section 4, the adaptive conversion algorithm based on the chain code contour reconstruction is discussed. An example is illustrated in Section 5. Analysis of the algorithm is given in Section 6. Finally, conclusions are made in Section 7.

2. Traversal algorithm for pointer-based quadtree

Samet [14] proposed a generic top-down quadtree traversal algorithm in which each node and all 8 neighbors (orthogonally and diagonally adjacent) are visited in preorder. Fuhrmann [13] proposed a simple variation of Samet's algorithm which can be described in the C language style as follows.

```
struct NODE {
    char color;
    char NODE *nw, *ne, *se, *sw;
};

traverse (root, w, n, e, s, nwc, nec, sec, swc)
struct NODE *root, *w, *n, *e, *nwc, *nec, *sec, *swc;
{
  if (root → color = = GRAY) {
    /* traverse the northwest son of root */
    traverse (root → nw, w → ne, n → s w, root → ne, ro
      ot → sw,
        nwc → se, n → se, root → se, w → se);

    /* traverse the northeast son of root */
    traverse (root → ne, root → nw, n → se, e → nw,
      root → se,
        n → sw, nec → sw, e → sw, root → sw);

    /* traverse the southwest son of root */
    traverse (root → sw, w → se, root → nw, root → se,
      s → nw
```

```
            w → ne, root → ne, s → ne, swc → ne);

        /* traverse the southeast son of root */
        traverse (root → se, root → sw, root → ne, e → sw,
          s → ne
            root → nw, e → nw, sec → nw, s → nw);
    }
    else
        function call( );
    }
```

This algorithm descends the quadtree by recursively calling the function of traverse with nine parameters which are the current node pointer (*root*) and the pointers of its eight neighbors (*w*, *n*, *e*, *s*, *nwc*, *nec*, *sec*, *swc*; where *w* is west, *n* is north, *e* is east, *s* is south and *c* is mnemonic for corner). Once the traversal step reaches the leaf node, the procedure *functioncall*( ) is performed. The tracking for the 8 neighbors of the SE child of the most north-west quadrant in a 64 × 64 image is shown in Table 1. Here, we modify the quadtree traversal algorithm [14] to be suited for the chain coding process.

```
/* return a chain set */
CHAIN *traverse (
        /* the left upper corner and the right lower corner
          coordinates */
    int $x_1$, $y_1$, $x_2$, $y_2$,
        /* parent, pseudo-parent and child type */
        ptype, pptype, ctype,
        /* grand parent node's color, current level and
          tree level */
        grandparentcolor, current_level, tree_level,
        /* current root node and its 8-neighbors */
    NODE root, w, n, e, s, nwc, nec, sec, swc)
{
  /* new node pointers for the next traverse */
  NODE *root p, *wp, *np, *ep, *nwcp, *necp, *secp,
    *swcp;
  /* chain sets for the four children of the current root
    node
      and the resulting chain set from the merging or
        chain coding. */
CHAIN *link_nw, *link_ne, *link_sw, *link_se, *flink;
/* If it is a white node, no need to go further */
if (root → color = = WHITE)
  return (DUMMY_EMPTY_CHAIN);


/* If the current level is not equal to the tree level,
  continue
  the traverse */
  if
    ((ptype = = NW_TYPE || pptype = = NW_TYPE)
    && ctype = = SW_TYPE)
    pptype = NW_TYPE;
  else if
    ((ptype = = SW_TYPE || pptype = = SW_TYPE)
    && ctype  = = NW_TYPE)
    pptype = SW_TYPE;
  else if
    ((ptype = = NE_TYPE || pptype = = NE_TYPE)
    && ctype = = SE_TYPE)
    pptype = NE_TYPE;
  else if
    ((ptype = = SE_TYPE || pptype = = SE_TYPE)
    && ctype  = =  NE_TYPE)
    pptype = SE_TYPE;
  else
    pptype = ctype;

  /* traverse the NW child */
  if (gpcolor = = BLACK && ctype = = SE_TYPE) {
    /* If its grandparent is a black node, its parent is
      SE_TYPE,
      and it is a NW child, then no contour will be
        generated.
      No need to go further */
    link_nw = DUMMY__EMPTY__CHAIN;
  }
  else {
    /* assign new node pointers for the NW child */
```

Table 1
The tracking of the traversal alogrithm for the 8-neighbors of the SE child of the most northwest quadrant in a 32 × 32 image

Neighbor type	1st stage	2nd stage	3rd stage	4th stage
Central pixel	root → nw	root → nw → nw	root → nw → nw → nw	root → nw → nw → nw → se
West	Null	Null	Null	root → nw → nw → nw → sw
North	Null	Null	Null	root → nw → nw → nw → ne
East	root → nw	root → nw → ne	root → nw → nw → ne	root → nw → nw → ne → sw
South	root → sw	root → nw → sw	root → nw → nw → sw	root → nw → nw → sw → ne
NW corner	Null	Null	Null	root → nw → nw → nw → nw
NE corner	Null	Null	Null	root → nw → nw → ne → nw
SE corner	root → se	root → nw → se	root → nw → nw → se	root → nw → nw → se → nw
SW corner	Null	Null	Null	root → nw → nw → sw → nw

```
    nw_child_assign (root, w, n, e, s, nwc, nec, sec, swc,
      &rootp, &wp, &np, &ep, &sp, &nwcp, &necp,
        &secp, &swcp);

    link_nw = traverse ($x_1, y_1, (x_1 + x_2)/2, (y_1 + y_2)/2$,
        ctype, pptype, NW_TYPE,
        current_level + 1, tree_level,
        rootp, wp, np, ep, sp, nwcp, necp, secp, swcp);
  }
  /* traverse the NE child */
  if (gpcolor = = BLACK && ctype = = SW_TYPE) {
    link_ne = DUMMY + EMPTY__CHAIN;
  }
  else {
    ne_child_assign (root, w, n, e, s, nwc, nec, sec, swc,
      &rootp, &wp, &np, &ep, &sp, &nwcp, &necp,
        &secp, &swcp);
      link_ne = traverse ($(x_1 + x_2)/2, y_1, x_2(y_1 + y_2)/2$,
        ctype, pptype, NE_TYPE,
        current_level + 1, tree_level,
        rootp, wp, np, ep, sp, nwcp, necp, secp, swcp);
  }

  /* traverse the SW child */
  if (gpcolor = = BLACK && ctype = = NE_TYPE)
    link_sw = DUMMY__EMPTY__CHAIN;
  else {
    sw_child_assign (root, w, n, e, s, nwc, nec, sec, swc,
      &rootp, &wp, &np, &ep, &sp, &nwcp, &necp,
        &secp, &swcp);
  link_sw = traverse ($x_1, (y_1 + y_2)/2, (x_1 + x_2)/2, y_2$,
        ctype, pptype, SW_TYPE,
        current_level + 1, tree_level,
        rootp, wp, np, ep, sp, nwcp, necp, secp, swcp);
  }
  /* traverse the SW child */
  if (gpcolor = = BLACK && ctype = = NE_TYPE) {
    link_sw = DUMMY__EMPTY__CHAIN;
  }
  else {
    se_child_assign(root, w, n, e, s, nwc, nec, swc, sec,
      &rootp, &wp, &np, &ep, &sp, &nwcp, &necp,
        &secp, &swcp);
    link_se = traverse ($(x_1 + x_2)/2, (y_1 + y_2)/2, x_2, y_2$,
        ctype, pptype, SE_TYPE,
        current_level + 1, tree_level,
        rootp, wp, np, ep, sp, nwcp, necp, secp, swcp);
  }
  /* merge the chain sets from the four children */
  flink = merge (link_nw, link_ne, link_sw, link_se);
}
else {
  /* perform the chain coding */
  flink = chain_coding (root, w, n, e, s, nwc, nec, sec, swc,
    $x_2, y_2$);
}
  return (flink);
}

/* follow Fuhrmann's traverse algorithm to assign
  8-neighbor
  to the NW child */
void nw_child-assign (
      /* input nodes */
  NODE *root, *w, *n, *e, *s, *nwc, *nec, *sec, *swc,
      /* output nodes */
      *rootp, *wp, *np, *ep, *sp, *nwcp, *necp, *secp,
        *swcp;
{
  if (root → color ! = GRAY)
    *rootp = *ep = *sp = *secp = root;
  else {
    *rootp = root → nw;
    *ep = root → ne;
    *sp = root → sw;
    *secp = root → se;
  }

  if (w = = NULL)
    *wp = *swcp = DUMMY_WHITE_NODE;
  else {
    if (w → color ! = GRAY)
      *wp = *swcp = w;
    else {
      *wp = w → ne;
      *swcp = w → se;
    }
  }

  if (n = = NULL)
    *np = *necp = DUMMY_WHITE_NODE;
  else {
    if (n → color ! = GRAY)
      *np = *necp = n;
    else {
      *np = n → sw;
      *necp = n → se;
    }
  }

  if (nwc = = NULL)
    *nwcp = DUMMY_WHITE_NODE;
  else {
    if (nwc → color ! = GRAY)
      *nwcp = nwc;
    else
      *nwcp = nec → se;
  }
}
```

Our method uses "virtual" leaves of the tree by "pretending" that the child of a leaf is a leaf of the same color. Even though a node does not have a neighbor, the algorithm provides a virtual neighbor which preserves the correct color. For example, a leaf node which represents a pixel in the southern boundary of an image has no south neighbor-hood. This traversal algorithm will pretend that it has "white" colored neighbors in *s*, *swc* and *sec*. The assignment for a node's 8 neighbors is done in the *xx_child_assign* procedure, where *xx* denotes *nw*, *ne*, *sw* and *se*. If the color of a neighbor node is not gray, we use the same node instead for being no descendant of that node. During the traverse, if it is a white node, the further traverse will be skipped for being no contour generation. If it is a black node and its grandparent's color is black, then this black node may not generate a contour. The criteria are quite simple and described as follows:

1. The black node is NW_TYPE and its parent is SE_TYPE.
2. The black node is NE_TYPE and its parent is SW_TYPE.
3. The black node is SW_TYPE and its parent is NE_TYPE.
4. The black node is SE_TYPE and its parent is NW_TYPE.

Therefore, this algorithm will only traverse the black nodes which lie on the boundary of objects. When the traverse reaches the deepest tree level, the *chain_coding* procedure is performed. The location (coordinates) of the pixel is computed and passed down from each traversal step. The X coordinate is equal to x_2 and the Y coordinate is equal to y_2. The coordinates start from left to right and top to bottom. The coding procedure is performed only on the leaf nodes including all virtual leaf nodes of the quadtree at the tree level (deepest one). For any other node, the *merging* procedure is performed. The input of the *merging* procedure is the coding result or the merging result of the subtree of that node.

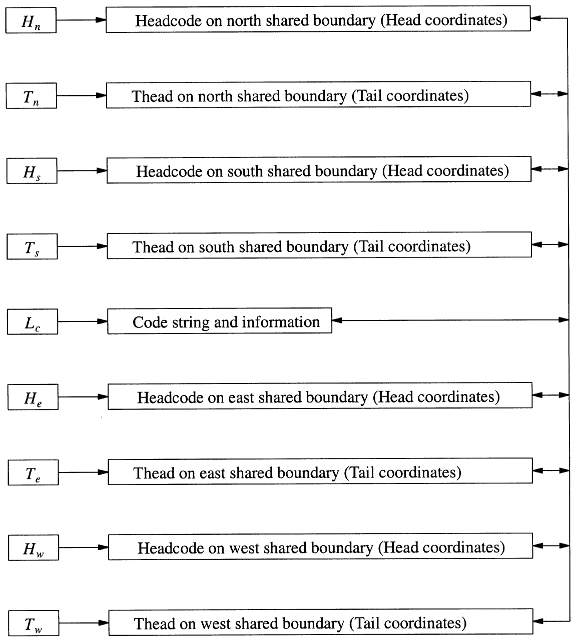
3. Chain coding for quadtree traversal algorithm

To help readers understand the proposed chain coding algorithm, we first briefly describe our previous work [6,7,15]. Let "Headcode" denote the first code of a chain and let "Thead" denote the code which concatenates with the tail of a chain. Let "Head coordinates" denote the location where the first code points to and let "Tail coordinates" denote the location where the Thead stands on. Let "codelink" be a chain code created for an object pixel. Codelinks will join together in the merging processing and turn to a chain. A chain set is composed of chains.

A set of 3×3 masks is able to contain all varieties of chain codes in the neighborhood of an object pixel. According to the permutation, a look-up table is set up for the code generation. During the scanning of the input image, if an object pixel is visited, a weighted window is convolved with the 3×3 neighborhood centered at that pixel. This 3×3 window, which is incorporated with different weights at each element exploring the presence of eight neighboring locations, is used to calculate the index value of the look-up table. Then, a series of operations are applied to concatenate these individual codelinks with an existing chain or to create a new chain. The conditions for the concatenation are based on the matching of the coordinates and the chain codes of the joining point.

To convert from the quadtree to chain code, the link-list data structures which are convenient in the merging process are used. The nine link-list data structures in Fig. 2 are used to describe a chain set. There are four shared boundaries for a sub-image in a quadrant: north, south, east and west boundaries. Link-lists H_n, H_s, H_e and H_w store the Head coordinates and the Headcode of a chain. Link-lists T_n, T_s, T_e, and H_w store the Tail coordinates and the Thead of a chain. The subscripts n, s, e and w denote the north (upper), south (lower), east (right), and west (left) boundaries. The last link-list L_e is for the code strings and information. The subscript c denotes the chain.

When a codelink is created, we must find out which shared boundary its Head and Tail are located at. Later, the merging procedure can be performed correctly. When the codelink is 0, it points to the east direction. It will connect to a chain whose Thead is located at its eastern shared boundary. Therefore, its Head coordinates of the



H_n
Headcode on north shared boundary (Head coordinates)
T_n
Thead on north shared boundary (Tail coordinates)
H_s
Headcode on south shared boundary (Head coordinates)
T_s
Thead on south shared boundary (Tail coordinates)
L_c
Code string and information
H_e
Headcode on east shared boundary (Head coordinates)
T_e
Thead on east shared boundary (Tail coordinates)
H_w
Headcode on west shared boundary (Head coordinates)
T_w
Thead on west shared boundary (Tail coordinates)

Fig. 2. The link-list data structure for a chain set.

codelink are added to H_e. When the codelink is 4, it points to the west direction. It will connect to a chain whose Thead is located at its western shared boundary. Therefore, its Head coordinates are added to H_w. From the similar concept, if the codelink is 2, we add them to H_n. If the codelink is 6, we add them to H_s. When the codelink is 1, 3, 5, or 7, it points to the northeast, northwest, southwest and southeast direction, respectively. The shared boundary type where the connection occurs will depend on its pixel location. Before an example illustration, we need to describe the order in the merging procedure to make it easier to understand the assignment of the Head coordinates. As aforementioned, a non-leaf node has four children. When a non-leaf node is merged, we first merge the sub-trees of its *nw* and *ne* children. The merging direction is the x-direction. Then, it follows by the merging the sub-trees of its *sw* and *se* children. The merging direction is the y-direction.

Table 2
The generation rules of the pseudo-parent type of a node

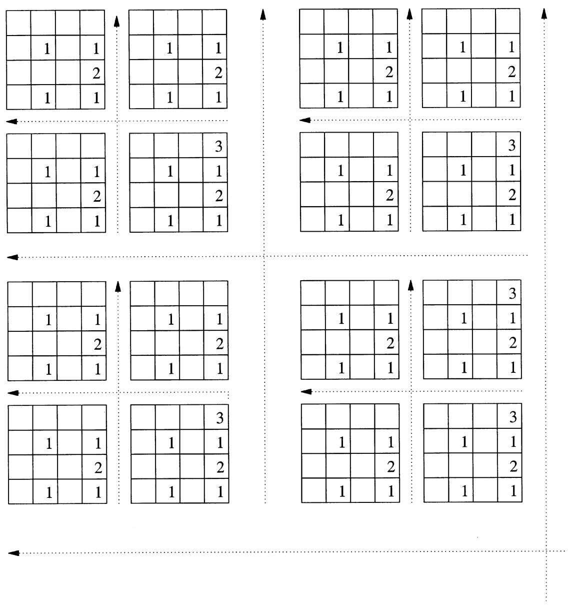
Parent type or pseudo-parent type	Node type	Pseudo-parent type passed down to child
NW	SW	NW
NE	SE	NE
SW	NW	SW
SE	NE	SE

If the codelink is 1, it points to the northeast direction. If the pixel which generates the codelink is a SE child in a quadrant, the codelink will connect on the eastern shared boundary. Therefore, its Head coordinates are added to H_e. If the pixel which generates the codelink is a NW or SW child in a quadrant, the codelink will connect on the northern shared boundary. Therefore, its Head coordinates are added to H_n. If the pixel which generates the codelink is a NE child in a quadrant, the codelink will connect whether on the eastern shared boundary or the northern boundary. The exact shared boundary type where the connection will occur depends on the pairs which are the parent type or the pseudo parent type and the type of the node. The pseudo-parent type is an inheritance based on some special combinations of the node and its parent. For example, if the type of a node is SW and that of its parent is NE but its pseudo-parent type is NW, then it will pass down the parent type SW and pseudo-parent type NW to its child. If the situation is not in these rules, the pseudo-parent type is equal to the parent type. These rules are shown in Table 2. From the similar deduction, the shared boundaries where the Tail of codelinks are located are classified. Both of them are shown in Table 3. Codelink "1" is used as an example to illustrate the classification in Fig. 3. In the figure, we only show the NW portion (16×16) of a 32×32 input image. All other cases can be derived by the same concept.

The order of the codelink creation on a pixel has been shown in Fig. 4. The order granted to each codelink

Table 3
Classification of the shared boundary occurrence for the Head and Tail of a codelink

Headcode; Thead	Child type in a quadrant	Parent or pseudo-parent type	Add to shared boundary
0; 0	NW, NE, SW, SE	Don't care	H_e; T_w
1; 5	NW, SW	Don't care	H_n; T_n
	NE	NW, NE, SW	H_n; T_n
	NE	SE	H_e; T_e
	SE	Don't care	H_e; T_e
2; 2	NW, SW, SW, SE	Don't care	H_n; T_s
3; 7	NE, SE	Don't care	H_n; T_n
	NW	NW, NE, SE	H_n; T_n
	NW	SW	H_w; T_w
	SW	Don't care	H_w; T_w
4; 4	NW, NE, SW, SE	Don't care	H_w; T_e
5; 1	NE, SE	Don't care	H_s; T_s
	SW	NE, SW, SE	H_s; T_s
	SW	NW	H_w; T_w
	NW	Don't care	H_w; T_w
6; 6	NW, NE, SW, SE	Don't care	H_s; T_n
7; 3	NW, SW	Don't care	H_s; T_s
	SE	NW, SW, SE	H_s; T_s
	SE	NE	H_e; T_e
	NE	Don't care	H_e; T_e



1: derived from the node type which is equal to SE.
2: derived from the node type which is NE and its parent type which is SE
3: derived from the node type which is NE and its pseudo parent type which is SE

if a node is marked 1, 2 or 3 and the codelink is 1, the head of codelink will be He
Otherwise, it will be Hn.

Fig. 3. An example to illustrate the classification of the location for the Head and Tail of codelink 1.

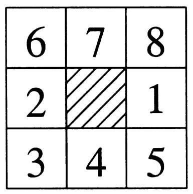


Fig. 4. The order of the codelink generation.

generated on a pixel will make Head and Tail in order in the link list as many as it could be. From our observation, some particular cases do not produce the resulting H_e, H_w, T_n or T_s in order. An extra swapping procedure is needed for the adjustment. The swapping procedure is to swap the two related Heads or Tails after the codelinks are created. It restores the sequence to make it in order. H_e or H_w will not be in order when a pixel generates two codelinks and both Heads are on the same link list H_e or H_w. The reason is that we grant the creation of codelinks 5 and 7 with the higher priority than 1 and 3. When codelinks 5 and 3 are assigned to H_w, or 7 and 1 are assigned to H_e for a pixel, the sequence of the Head coordinates will not be in order in the vertical direction. Therefore, the swapping procedure is necessary. We show all the exception cases in this category in

Table 4
The particular cases of the two Heads generated on a pixel will not be in order

Headcode	Child type in a quadrant	Parent or pseudo-parent type
3 & 5	NW	SW
	SW	NW
1 & 7	NE	SE
	SE	NE

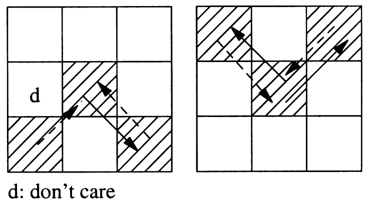


Fig. 5. The patterns that need the extra swapping procedure.

Table 4. The other category is for T_n and T_s. The patterns which need the extra swapping on T_n or T_s are shown in Fig. 5.

By using this approach, the Head and Tail coordinates of the open chains are in sequence in the link-list structure. When the quadtree traversal algorithm in Section 2 reaches the leaf nodes, all 8 neighbors are known. From their color we can calculate the index value of the look-up table for that pixel (leaf node). Then the chain codes are extracted and placed into a chain set. When the chain coding procedures for all the children of the same parent are finished, we merge the chain sets together. Basically, when two subtrees, named A and B, are merged together, we will merge the Head of subtree A to that of B. Then it follows by the Tail of subtree A to that of B. If a chain is closed after merging, it is added into the chain set C of the link-list L_c. Otherwise, the open chain is added into the chain set O of the link-list L_c. The same process is performed recursively from the leaf nodes back to the root node.

4. Adaptive level-based chain coding retrieval

As aforementioned, the Boolean operation of two quadtrees is the fundamental operation for GIS. If we need the chain code information of the resulting quadtree, we must perform the chain coding procedure for the new constructed quadtree. If we compare the new quadtree with the old one, it may not have great difference. This task can be simplified by constructing the chain code information of the new quadtree from that of the old one.

The link-list data structure used to describe the subtree of a node is simple and easy to maintain. During the merging procedure, we can save the intermediate result of all the nodes which are smaller than a given level. This information will be retrieved when we want to find the chain code contour of the new constructed quadtree. This step will avoid lots of unnecessary re-coding processes. During the Boolean operation of two quadtrees, if the color of a node in the original quadtree is changed, we will mark it and all its ancestors are changed. After the Boolean operation is finished, we perform the chain coding algorithm. If a node is not marked, we need not go further. Instead, we retrieve the link-list information from what we have saved before. If a node is marked, we track its four children to see whether any of them is marked. This process is performed recursively until the leaf node is reached or all the four children of the node are not marked.

Since it is impossible to save all the nodes' information, we can save a portion of nodes instead. By given a level number, we can save all the information of nodes which are less than or equal to that level number. During the chain coding of the new quadtree, if a node is not marked and its level number is less than or equal to the given level, we can retrieve their chain codes directly. If its level number is larger than the given level, we must track all its descendant. This approach is adaptive. It the hardware has plenty of memory, we should use a small level number. Otherwise we use a big number.

5. Experimental results

We use Fig. 1 as an example to illustrate to apply our conversion algorithm. Recall that the chain coding process is performed on the leaf nodes or the virtue leaf nodes in a quadtree. The result of the merging process for each internal node and leaf with black color is shown in Table 5. C, E, F, J and K are leaves with black color but only J and K are individual pixels in the input image. B, D, G, H and I are internal nodes. Root node is marked as A which has two closed chains. They are represented as (0, 1) 0246 and (3,4) 444666000001044432102465, where (x, y) are column and row coordinates of the starting pixel in the chain code description and the numbers follow denote the code sequence of the contour. The code sequence is read from left to right.

6. Analysis

Let the size of an image be $N \times N$ and the number of boundary pixels be $O(B)$. The height of the quadtree (including virtual leaves) is H, where $N = 2^H$. Each chain coding and merging operation costs time $O(1)$. The time requiring merging of the chains of two quadrants is dependent on the number of joints in these two quadrants. There is no need to sort or search for the two pairs of the joint (the boundary pixel) since the Heads and Tails are in order. In the sequential mode, the total number of joints in the image is in the same order of boundary pixels and that is the time needed in the traverse of nodes. The complexity of our algorithm is dependent on the height of the traverse plus the operation of the boundary pixels. Therefore, it is $O(H + B)$.

For a pyramid architecture with $O(N)$ processors and each processor handling a subquadrant from its parent, the traverse of the tree needs time $O(H)$ in parallel. The chain coding operation is $O(1)$. The merging operation in a processor of level l requires time which is in the same order of the number of open chains. The worst case occurs when the order open chains' number is equal to the order of the side's length of the quadrant in level l. If

Table 5
The result of the merging process for each node in Fig. 5

Node	Head	Tail	Starting pixel	Chain code sequence
C			(0, 1)	0246
B			(0, 1)	0246
E	H_s (3, 4)	T_s (4, 3)		02465
D	H_s (3, 4)	T_s (4, 3)		02465
F	H_e (4, 7)	T_n (3, 4)		444666000
	H_n (4, 3)	T_e (3, 5)		21
H	H_w (3, 5)	T_e (5, 6)		43
	H_e (6, 6)	T_w (4, 7)		01
J	H_e (7, 6)	T_w (6, 6)		0
	H_w (5, 6)	T_e (6, 6)		4
K	H_w (6, 6)	T_w (7, 6)		4
I	H_w (5, 6)	T_w (6, 6)		044
G	H_w (3, 5)	T_w (4, 7)		0104443
A			(0, 1)	0246
			(3, 4)	444666000001044432102465

the number of the pixels of a quadrant in level l is $n \times n$, the time for the merging operation will require $O(n)$. If we sum up the merging time for all levels in parallel, which is $2^H + 2^{H-1} + \cdots + 2^2$, it will be $O(2^H)$ or $O(N)$.

7. Conclusion

In this paper, an adaptive algorithm for converting the quadtree representation to the chain code representation is presented. Different from other algorithms which need to search for the neighbors of the current node, the algorithm is based on traversing the quadtree. Therefore, it is recursive and has the parallelism. It can be easily implemented in a pyramid architecture. Besides, we introduce an adaptive method to construct the chain codes for the resulting quadtree of the Boolean operation of two quadtrees by re-using the original chain codes. This adaptive method is quite useful and significantly speeds up the conversion in all kinds of applications.

References

[1] H. Samet, Applications of Spatial Data Structures — Computer Graphics, Image Processing, and GIS, Addison-Wesley, New York, 1990.

[2] T-W. Lin, Set operations on constant bit-length linear quadtrees, Pattern Recognition 30 (7) (1997) 1239–1249.

[3] F. Dehne, A. Rau-Chaplin, A.G. Ferreira, Hypercube algorithms for parallel processing of pointer-based quadtree, Comput. Vision Image Understanding 62 (1) (1995) 1–10.

[4] Y. Hung, A. Rosenfeld, Parallel processing of linear quadtree on a mesh-connected computer, J. Parallel Distrib. Comput. 7 (1989) 1–27.

[5] H. Freeman, Computer processing of line drawing images, Comput. Surveys 6 (1) (1974) 57–97.

[6] F.Y. Shih, W-T. Wong, A new single-pass algorithm for extracting the mid-crack codes of multiple regions, J. Visual Commun. Image Rep. 3 (3) (1992) 217–224.

[7] W-T. Wong, Y-L. Chen, F.Y. Shih, A fully parallel algorithm for the extraction of chain and mid-crack codes of multiple contours, Conference Proceedings of International Computer Symposium, HsinChu, Taiwan, R.O.C., 1994, pp. 565–570.

[8] P. Zingaretti, M. Gasparroni, L. Vecci, Fast Chain Coding of Region Boundaries, IEEE Trans. Pattern Anal. Mach. Intell. 20 (4) (1998) 407–415.

[9] H. Samet, Region Representation: quadtree from boundary codes, Commun. ACM 23 (3) (1980) 163–170.

[10] M.R. Lattanzi, C.A. Shaffer, An optimal boundary to quadtree conversion algorithm, CVGIP: Image Understanding 53 (3) (1991) 303–312.

[11] C.R. Dyer, A. Rosenfeld, H. Samet, Region representation: boundary codes from quadtrees, Commun. ACM 23 (3) (1980) 171–179.

[12] G.N. Kumar, N. Nandhakumar, Efficient object contour tracing in a quadtree encoded image, SPIE Appl. Artif. Intell. IX 1468 (1991).

[13] D.R. Fuhrmann, Quadtree traversal algorithms for pointer-based and depth-first representation, IEEE Trans. Pattern Anal. Mach. Intell. 10 (6) (1988) 955–960.

[14] H. Samet, A top-down quadtree traversal algorithm, IEEE Trans. Pattern Anal. Mach. Intell. (T-PAMI) 7 (1) (1985) 94–98.

[15] F.Y. Shih, W-T. Wong, A one-pass algorithm for local symmetry of contours from chain code, Pattern Recognition 32 (7) (1999) 1203–1210.

About the Author—FRANK Y. SHIH received the B.S. degree from National Cheng-Kung University, Taiwan, in 1980, the M.S. degree from the State University of New York at Stony Brook, in 1984, and the Ph.D. degree from Purdue University, West Lafayette, Indiana, in 1987, all in electrical engineering. He is presently a professor jointly appointed in the Department of Computer and Information Science (CIS) and the Department of Electrical and Computer Engineering (ECE) at New Jersey Institute of Technology, Newark, New Jersey. He currently serves as an associate chairman of the CIS department and the director of Computer Vision Laboratory. Dr. Shih is on the Editorial Board of the International Journal of Systems Integration. He is also an associate editor of the International Journal of Information Sciences, and of the International Journal of Pattern Recognition. He has served as a member of several organizing committees for technical conferences and workshops. He was the recipient of the Research Initiation Award from the National Science Foundation in 1991. He was the recipient of the Winner of the International Pattern Recognition Society Award for Outstanding Paper Contribution. He has received several awards for distinguished research at New Jersey Institute of Technology. He has served several times in the Proposal Review panel of the National Science Foundation on Computer Vision and Machine Intelligence. He holds the IEEE senior membership. Dr. Shih has published over 110 technical papers in well-known prestigious journals and conferences. His current research interests include image processing, computer vision, computer graphics, artificial intelligence, expert systems, robotics, computer architecture, fuzzy logic, and neural networks.

About the Author—WAI-TAK-WONG received the B.S. degree in Chemical Engineering from National Taiwan University in 1986. He received the M.S. degree and the Ph.D. degree in Computer and Information Science from New Jersey Institute of Technology in 1992 and 1999. He has worked for Syncsort Incorporate as a Software Engineer to develop enterprise level heterogeneous client server computing software since 1995. He received the Pattern Recognition Society Award in 1995. His current research interests are parallel algorithms, image processing, client server computing, distributed computing and software engineering.