ACADEMIC PRESS

Available at www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT®

J. Vis. Commun. Image R. 14 (2003) 97–113

JOURNAL OF VISUAL Communication & IMAGE Representation

www.elsevier.com/locate/yjvci

New two-phase spatial data structures with applications to binary images

Kuo-Liang Chung,* Hsu-Lien Huang, and I-Chien Chen

Department of Computer Science and Information Engineering, National Taiwan University of Science and Technology, No. 43, Section 4, Keelung Road, Taipei 10672, Taiwan, ROC

Received 25 August 2000; accepted 27 February 2003

Abstract

Considering a binary image, a new two-phase representation is presented in this paper to reduce the memory requirement in the conventional tree-based spatial data structures (SDSs) such as the linear quadtree, DF-expression, S-tree representation, etc. Experimental results show that not only our proposed two-phase representation has a better memory-saving effect but it also can speed up the coding-time when compared to the existing SDSs. We also show that our proposed two-phase representation has a better computational performance when running geometric operations, such as computing the area and the centroid, on the proposed two-phase representation directly.

Keywords: Connected component; Geometric operations; Image representation; Spatial data structures

1. Introduction

Quadtree is the most well-known spatial data structure (SDS) for representing binary images and can reduce the memory requirement through the use of aggregation of homogeneous blocks (Samet, 1990b). Based on different kinds of SDSs, Samet (1990a) gave an excellent survey on many applications in computer graphics, image processing, geographic information systems, image database, pattern recognition, etc.

* Corresponding author. Fax: +886-2-273-01081.
E-mail address: klchung@cs.ntust.edu.tw (K.-L. Chung).

1047-3203/03/$ - see front matter
doi:10.1016/S1047-3203(03)00021-X

In order to reduce the storage requirement, some memory-saving representations have been proposed. Gargantini (1982) presented a pointerless SDS, called the linear quadtree (LQ), which represents a quadtree by encoding only black nodes, and each black node is encoded by a path from the root to the node. The DF-expression (Kawaguchi and Endo, 1980) encodes each node of the quadtree in depth first search (DFS) manner using the symbol 'G,' 'B,' or 'W' to indicate the gray node, black node or white node, respectively. Bintree is another storage-efficiency SDS representation. Based on the bintree, Jonge et al. (1994) presented the S-tree representation. The S-tree is obtained by traversing the bintree in DFS manner using 1 (0) to encode the external node (internal node) and then using 1 (0) to encode the black (white) external node. The bincodes representation proposed by Ouksel and Yaagoub (1992) were shown to have some space improvement over the LQ in empirical comparisons (Shaffer et al., 1993).

Given a binary image, this paper first gives an observation that the conventional SDSs mentioned above spend much amount of memory for storing leaves and the internal nodes near leaves when the binary image has some detailed texture. In order to reduce the memory requirement in the conventional SDSs, a new two-phase representation is presented in this paper. In the first phase, we follow the conventional tree-based SDS from root to a specific level, say s, of the tree. There exist three kinds of leaves at level s representing the entirely white subimages, the entirely black subimages, and the gray subimages. The third kind which we name it as the gray leaf is the most memory consuming part in the conventional SDSs. In the second phase, each gray leaf obtained from the first phase is coded by the connected component string (CCS) coding scheme. The CCS is obtained by employing the morphological technique (Serra, 1982) and can efficiently represent the gray leaf and it leads to the memory-saving effect. Experimental results show that our improved two-phase SDS over the conventional SDSs have 66.14, 19.49, 12.33, and 48.83% memory improvement ratios (12.69, 4.66, 2.49, and 18.71% coding-time improvement ratios) when compared to the LQ, DF-expression, S-tree representation, and the bincodes, respectively. Besides the memory-saving effect and the encoding-time speedup, we also show that our proposed two-phase representation has a better computational performance when running two geometric operations, computing the area and the centroid, on the proposed two-phase representation directly.

The remainder of this paper is organized as follows. Section 2 presents four kinds of conventional SDSs. Section 3 presents our proposed two-phase representation to improve the conventional SDSs. In Section 4, two geometric operations for computing the area and centroid on the proposed two-phase representation directly are discussed. Some experimental results are illustrated in Section 5. Finally, some conclusions are addressed in Section 6.

2. Conventional tree-based SDSs

In this section, we take a simple example to introduce the four existing tree-based SDSs mentioned above.

For a quadtree, if the images is entirely black (white), then the root node is labeled with 1 (0). Otherwise, the root node is further divided into four equal-sized subimages. For the quadrants, they are labeled *sw* (southwest), *se* (southeast), *nw* (northwest), and *ne* (northeast), respectively. The quadtree decomposition is based on repeatedly subdividing the subimages until the subimage is entirely black or white. Given a binary image with 8×8 as shown in Fig. 1, the corresponding quadtree is shown in Fig. 2.

2.1. Linear quadtree: LQ

Without using pointers, a LQ (Kawaguchi and Endo, 1980) represents the quadtree by a set of codes, and each code is obtained by encoding a path from the root to the black node, i.e., external node, in the quadtree. Let the *sw* quadrant, the *se* quadrant, the *nw* quadrant, and the *ne* quadrant be encoded with 0, 1, 2, and 3, respectively. Fig. 3 illustrates the assigned codes on each quadrant. The node *c* in Fig. 3 is encoded by 030 and the node *a* is encoded by $1X$ where X is a don't-care symbol. The don't-care symbol X denotes this node being not at the bottom level of the tree and its code is ended at X. By traversing the quadtree in DFS manner to code the black nodes, the quadtree in Fig. 3 is coded as 030 031 032 $1X$ $31X$ 330 331 333.

2.2. DF-expression

Given a quadtree, the DF-expression (Kawaguchi and Endo, 1980) is obtained by traversing the quadtree in DFS manner. The DF-expression is a sequence consisting of three symbols 'G,' 'W,' and 'B,' which denote the gray node, white node, and black node, respectively. During the traversal, if a gray node is encountered, the symbol 'G' is appended to the DF-expression; if a white node is encountered, the symbol 'W' is appended to the DF-expression, and the symbol 'B' is appended if a black node is encountered. For example, the DF-expression of Fig. 2 is *GGWWWGBBBW BWGWBWGBBWB*.

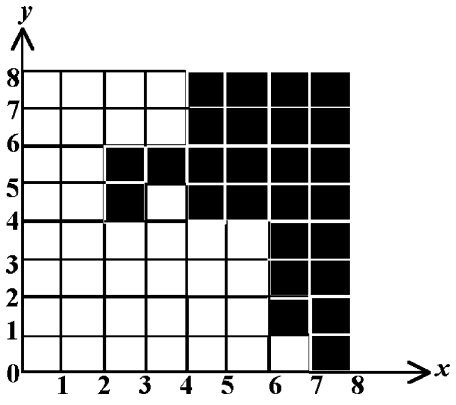

Fig. 1. A binary image example.

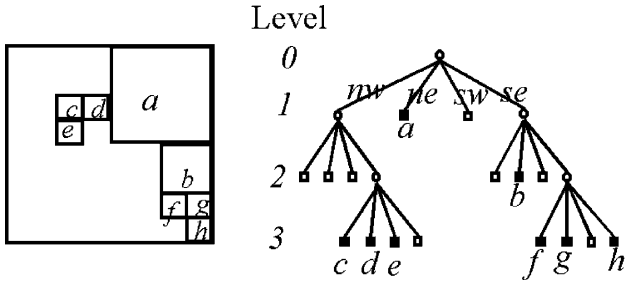

Fig. 2. The quadtree representation of Fig. 1.

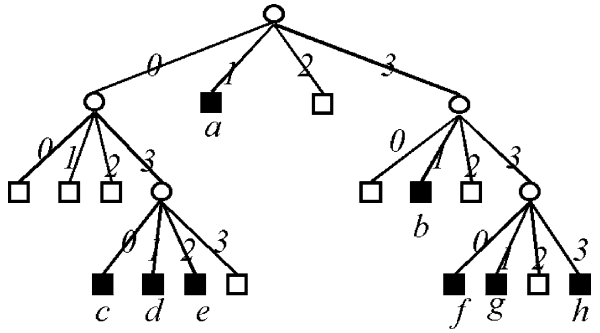

Fig. 3. The assigned digits of the LQ.

2.3. S-tree

The S-tree is based on the bintree (Samet, 1990b) structure. Given an image, the corresponding bintree is obtained by recursively subdividing the image into two equal-sized subimages until the subimage is totally black or white. At each step, the partition is alternated between the x- and y-axes. The corresponding bintree of Fig. 1 is shown in the right side of Fig. 4 and the partitioned subimages are shown in the left side of Fig. 4.

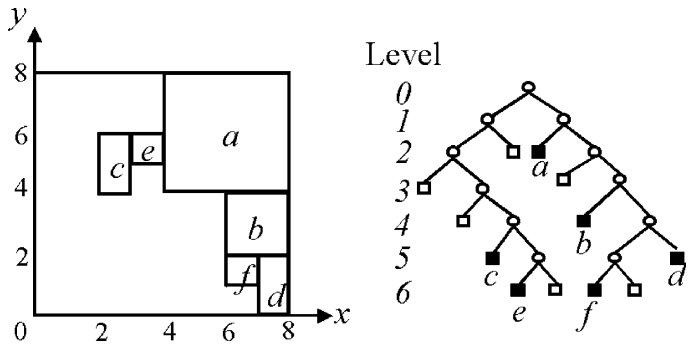

Fig. 4. The bintree representation of Fig. 1.

The S-tree representation is obtained by traversing the bintree in breadth-first search (BFS) manner and the traversed result is stored in two array tables, namely, the linear-tree table and the color table. While traversing the bintree, a '1' ('0') is emitted to the linear-tree table when an external (internal) node is encountered. Meanwhile, a '1' ('0') is emitted to the color table when a black (white) leaf node is encountered. For example, the S-tree representation for Fig. 4 is listed below:

linear-tree table: 0 00 0110 1010 1010 1001 1111
color table: 010001111010

2.4. Bincodes

Given an image, the representation of bincodes (Ouksel and Yaagoub, 1992) is obtained by traversing the corresponding bintree in DFS manner and encoding each black node by a location code. For a $2^N \times 2^N$ binary image, the representation of bincodes for a black node at location (x, y) and level l of its corresponding bintree is denotes $b(l, x, y)$. The location (x, y) is the coordinate of the left-bottom corner of the corresponding subimage. For example, the black node c in Fig. 4 denoted by $b(5, 2, 4)$ is at level 5 of the bintree and at the location (2,4) of the image. The encoding expression is $b(l, x, y) = \sum_{k=0}^{N-1}(x_k \times 2^{4k+3}) + \sum_{k=0}^{N-1}(y_k \times 2^{4k+1}) + \sum_{k=0}^{2N-1}(s_k \times 2^{2k})$, where $x_{N-1}x_{N-2}\cdots x_0$ and $y_{N-1}y_{N-2}\cdots y_0$ denote the binary representation of x and y, respectively, and $s = 2^{2N} - 2^{2N-l} = (s_{2N-1}s_{2N-2}\cdots s_0)_2$. In fact, the encoded bincode for the black node $b(l, x, y)$ is expressed as the sequence $(x_{N-1}s_{2N-1}y_{N-1}s_{2N-2}x_{N-2}s_{2N-3}\cdots x_0 s_1 y_0 s_0)_2$.

For the black node c in Fig. 4, we encoded as $b(5, 2, 4) = (011111010100)_2$, since $N = 3$, $x = (010)_2$, $y = (100)_2$, and $s = (111110)_2$. By traversing the bintree in DFS manner and encoding the black nodes, the bincodes representation of Fig. 4 is a strictly increasing ordered sequence as (2004, 2015, 3840, 3568, 3545, and 3548) in decimal representation.

3. Proposed two-phase representation

In this section, we first describe the idea of the proposed two-phase representation. Second, how to apply the morphological dilation operator to perform the connected component analysis is introduced. Then, the CCS coding scheme for representing the connected components is described. Finally, the formal algorithms for encoding and decoding the CCS are presented.

3.1. The idea

In the first phase, we adopt any one of the conventional SDSs to represent the given binary image. Instead of subdividing each subimage into an entirely black or white subimage, we stop the tree decomposition to a specified level to obtain an approximate quadtree or bintree for the image. Fig. 5 is an example of the approximate quadtree which is obtained by stopping the decomposition at level 1. The

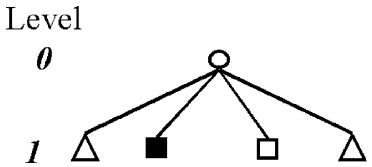

Fig. 5. The approximate quadtree of Fig. 1.

approximate tree has three kinds of leaves, namely, the black leaves, the white leaves, and the gray leaves. Each black leaf represents an entirely black subimage; each white leaf represents an entirely white subimage, and each gray leaf represents a gray subimage. The gray leaf should be a subtree in the conventional SDS. As described before, the gray leaves are the memory consuming part in the conventional SDSs. To reduce the memory requirement, in the second phase, we code the gray leaves using the connected component string (CCS) coding scheme. A CCS is a bit-stream which can efficiently represent a connected component in a subimage. The CCSs are obtained by employing the morphological technique in the connected components analysis for the subimages represented by gray leaves.

For example, the binary image shown in Fig. 1 can be represented by the quadtree in Fig. 2 using the conventional SDSs. Instead of decomposing the tree to level 3, we stop the decomposition at level 1 and obtain the approximate quadtree as in Fig. 5. The nodes marked as Δ's in Fig. 5 are the gray leaves which will be coded using the CCS coding scheme in the second phase.

3.2. Connected component analysis

To find a connected component for the subimage represented by a gray leaf, we employ the morphological dilation operation. Consider a set A to which the dilation operation will be associated with the structuring element B. Let $\oplus$ denote the morphological dilation operator. The dilated set $A \oplus B$ is defined to be the union of all pixels under the support of the structuring element B. An example of the dilation operation is shown in Fig. 6 where each square box in A and B is a pixel of the binary image.

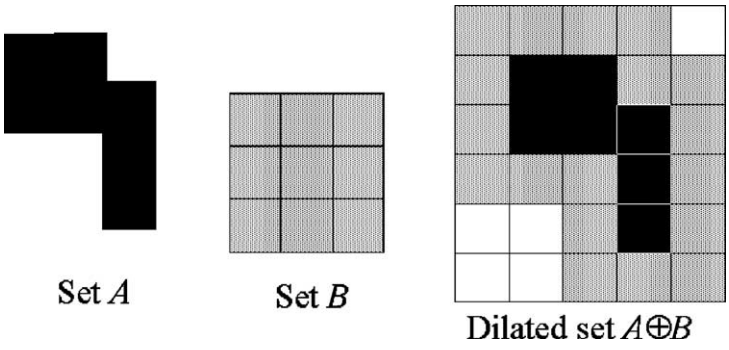

Fig. 6. The morphological dilation operation.

Let Y represent a connected component contained in the subimage G. Scanning G in a raster manner, we get a starting point of Y, say p. Then the following iterative expression (Gonzalez and Woods, 2002; Serra, 1982) yields all the pixels of Y

$$X_k = (X_{k-1} \oplus B) \cap G \quad \text{for } k = 1, 2, 3, \ldots, \tag{1}$$

where $X_0 = p$ and $\cap$ is the pixel-wise intersection. For each iteration, X_{k-1} extends to its connected neighbors by the shape of structuring element B within the subimage G. The iterations will stop at X_n when all the neighbors of X_n within G are visited. Here, the set X_n is indeed the the connected component Y contained in the subimage G. Since there may be more than one connected component in the subimage G, we can easily continue the raster scanning to find the starting point of the next connected component. Each connected component in the subimage G is represented by a starting point following by a connected component string (CCS). Previously, the morphological operations were also used in reducing the wavelet data in Chai et al. (1999). The next subsection describes how to generate the bit-stream CCS from Eq. (1).

3.3. The CCS coding scheme

The subimage G is corresponding to one gray leaf on the approximate tree. For preserving better geometric connectivity, the structuring element B used is 8-connected as shown in Fig. 6. When applying the dilation operation $X_{k-1} \oplus B$ in Eq. (1) for each pixel in X_{k-1}, there are eight neighbors of that pixel to be checked.

Initially, the set X_0 contains only one pixel, say p. To find its connected pixels, we start from the east neighbor of p and go clockwise to visit its eight neighbors. Upon visiting one neighbor, if this neighbor is not a black pixel, that means it is not a connected pixel, then we emit the bit '0' to the CCS. On the other hand, if this neighbor is a black pixel, that means it is a connected pixel, then the bit '1' is emitted to the CCS and then recursively go through the eight neighbors of this connected pixel.

An example of CCS coding simulation is illustrated in Fig. 7. In Fig. 7, the subimage G is of size 4×4 and its CCS is obtained by using five iterations. It is observed that the CCS needs only 10 bits and 4 bits for the location of the starting point, so totally 14 bits are required in this example. If this subimage is represented by a quadtree, there are 10 leaf nodes to be encoded. For Fig. 7, no matter how efficient the conventional tree-based SDS is, the memory required in these leaf nodes is relatively larger than that of the CCS coding scheme. This is the main reason why the proposed two-phase representation has a better memory-saving effect when compared to the conventional tree-based SDSs.

As described above, the proposed two-phase representation follows the conventional SDSs in the first phase to a specific tree level, then in the second phase, the CCS coding scheme is followed and has the memory-saving advantage when compared to the conventional tree-based SDSs. However, the concerning dilation operation in our method is time consuming for small subimages, such as for size 2×2 and 2×1. In order to improve the encoding and decoding time, the bit pattern of such a small subimage is recorded directly. For example, in Fig. 2, the most right

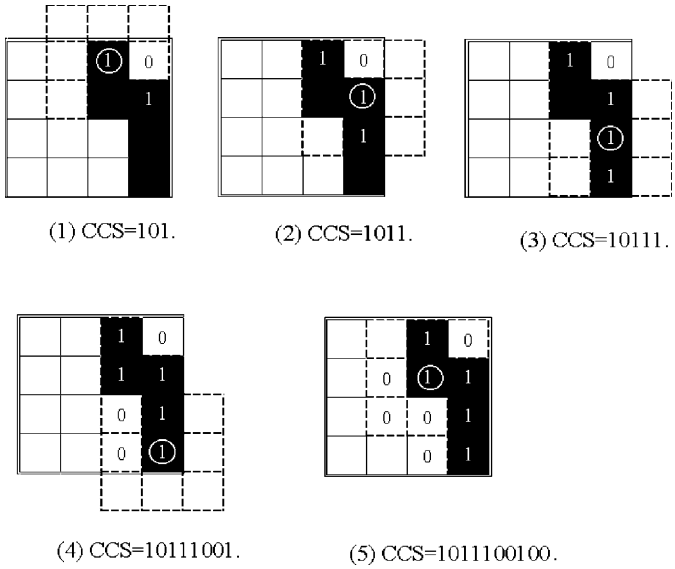

Fig. 7. CCS coding example.

subtree at level 2 represents a subimage of size 2×2. This 2×2 subimage has three black pixels, f, g, and h, and one white pixel. We record the bit pattern '1101' instead of iteratively performing the dilation operation.

3.4. The encoding and decoding procedures for CCS

In this subsection, the encoding and the decoding procedures for CCS are illustrated. As described before, the first phase can be any one of the conventional tree-based SDSs, their encoding (decoding) procedure can do as usual to the specific tree level and obtain the gray leaves of the approximate tree. Whenever a gray leaf is encountered, the CCS encoding (decoding) procedure is called to process the corresponding subimage G which is with respect to the gray leaf.

Initially, the location of the starting point p is given. As mentioned above, the p is obtained by scanning the subimage G in a raster manner. The subimage G is represented by a 2-D array *pixel*[,], each entry being BLACK or WHITE. In order to realize the morphological dilation operation in Eq. (1), eight-neighbors traversal technique is used. The **recursive procedure** *encode_CCS*(x, y) is recursively called for each visited pixel. The input parameter (x, y) indicates the coordinate of the current visited pixel. The structuring element B is represented by a 1-D array named *direction*[ ] with eight neighbors recording the clockwise visiting sequence. In addition, a 2-D boolean array named *visited*[,] indicates whether the corresponding pixel is visited or not.

recursive procedure *encode_CCS*(x, y);

```
/* The direction[ ] is a constant variable.*/
/* The pixel[ , ] and visited[ , ] are global variables.*/
/* The i, dx, and dy are local variables. */
  begin
    for i = 1 to 8 do /* Go through the eight neighbors. */
     begin
       (dx, dy) = direction[i]; /* Get the next neighbor's location. */
       if ((x + dx, y + dy) is within the subimage G) and
         (visited[x + dx, y + dy] = false) then
         set visited[x + dx, y + dy] to be true;
         if pixel[x + dx, y + dy] = BLACK then
           /* This is a connected pixel. */
           output the bit '1'; /* Emit bit '1' to CCS. */
           encode_CCS(x + dx, y + dy); /* Recursively calling the neighbor.*/
         else
           /* This is not a connected pixel. */
           output the bit '0'; /* Emit bit '0' to CCS. */
         end if;
       end if;
    end for;
  end procedure;
```

By the same arguments, the **recursive procedure** *decode_CCS*(x, y) is recursively called for each decoded pixel. Initially, the 2-D array *pixel*[,] representing the decoded subimage is set to be WHITE for all entries. Each decoded pixel will be set the value BLACK at the corresponding location in the array *pixel*[,].

```
recursive procedure decode_CCS(x, y);
/* The direction[ ] is a constant variable.*/
/* The pixel[ , ] and visited[ , ] are global variables.*/
/* The i, dx, dy, and bit are local variables. */
  begin
    for i = 1 to 8 do /* Go through the eight neighbors. */
     begin
       (dx, dy) = direction[i]; /* Get the next neighbor's location.*/
       if ((x + dx, y + dy) is within the subimage G)
         and (visited[x + dx, y + dy] = false) then
           set visited[x + dx, y + dy] to be true;
           bit = next_CCS(); /* Get the next bit in CCS. */
           if bit is '1' then
             /* This is a connected pixel. */
             pixel[x + dx, y + dy] = BLACK; /* Draw the pixel on the subim-
       age. */
             decode_CCS(x + dx, y + dy); /* Recursively calling the neighbor.
       */
           end if;
```

```
        end if;
    end for;
end procedure;
```

We now analyze the computational time complexity among the conventional SDSs and our proposed two-phase SDS. For simplicity, we take the quadtree-based SDSs, such as the LQ and the DF-expression, as the representative although the following time complexity analysis can also be applied to the binary tree-based SDSs, such as the S-tree and the bincodes. Since the built quadtree-like structures will dominate the encoding-time complexity, we thus focus on the time complexity for building up the quadtree-like structures by using the conventional one-phase method and our proposed two-phase method. Given an input image, let T_{one} and T_{two} be the time to build up the corresponding quadtree structure and the quadtree-like structure using the one-phase method and our proposed two-phase method, respectively. Suppose each subimage (with respect to the CCS coded subimage G) is of size $k \times k$ and the number of these subimages in the built quadtree-like structure using our two-phase method is m.

Considering the worst case, for each subimage G, it takes $O(k^2 \log k)$ time to build up the related subquadtree (Cormen et al., 2001) when using the one-phase method since the depth of the built subquadtree is $O(\log k)$ and at each level there are $O(k^2)$ pixels to be processed. Since there are m subimages G's to be considered, it takes $O(mk^2 \log k)$ time to build up these m corresponding subquadtrees. Therefore, $T_{\text{one}} = O(T' + mk^2 \log k)$, where T' denotes the time complexity required in building up the whole quadtree except those m subquadtrees with respect to the m subimages G's. For one subimage G with size $k \times k$, it takes $O(k^2)$ time to build up the bit-stream CCS using our proposed two-phase method since each pixel in G is scanned at most twice. Consequently, we have $T_{\text{two}} = O(T' + mk^2)$. Comparing T_{one} and T_{two}, it yields $T_{\text{one}} \geqslant T_{\text{two}}$. Besides the memory-saving effect, we thus claim that our proposed two-phase method has a faster encoding performance. In Section 5.1, the compression and encoding-time benefits of our proposed two-phase representation are demonstrated.

4. Two geometric operations

In this section, the geometric operations for calculating the area and centroid on the proposed two-phase representation are described. The area and the centroid (Gonzalez and Woods, 2002; Samet, 1990a) are two important regional descriptors. Especially, the centroid is often used in the central moments which are very useful in shape analysis.

Since the first phase is the same as the conventional SDSs, we only discuss the remaining operations for the CCS coded subimage.

4.1. Computing area

The area of a binary image is defined as the number of black pixels of the whole image. As described in Section 3, we emit a bit '1' whenever a black pixel is encoun-

tered in encoding the CCS. Therefore, the number of '1's in the CCS is the area of its corresponding connected component. Given a CCS of the subimage G, we can easily obtain the area of G by counting the number of '1's in the CCS. If there exists any other connected components in the subimage, the area of each connected component can be computed from its CCS and they are summed up together to obtain the area of G. For example, the subimage in Fig. 7, its CCS is '1011100100.' The number of '1's in the CCS is 5, so the area of this subimage is 5.

Following the same analysis technique as in Section 3.4, we still take the quadtree-based SDS as the representative. Suppose the quadtree structure and the quadtree-like structure using the one-phase method and our two-phase method, respectively, have been built up. From Sections 3.4 and 2, it is clear that in the worst case, there are $O(mk^2 \log k)$ nodes in those m subquadtrees to be considered when computing the area on the one-phase representation, but there are only $O(mk^2)$ nodes in those m subquadtrees to be considered when computing the area on our two-phase representation. Given an input image, let T^g_{one} and T^g_{two} be the time to compute the area on the corresponding quadtree structure and the quadtree-like structure using the one-phase method and our proposed two-phase method, respectively. We thus have that $T^g_{\text{one}} = O(T_g + mk^2 \log k)$, where T_g denotes the time complexity required in scanning the whole quadtree except those m subquadtrees with respect to the m subimages G's in order to accumulate the temporary results of the area. On the contrary, we have $T^g_{\text{two}} = O(T_g + mk^2)$. Since $T^g_{\text{one}} \geqslant T^g_{\text{two}}$, we claim that computing the area on our proposed two-phase representation described is faster than that on the one-phase representation.

4.2. Computing centroid

Assume a binary image is of size $2^N \times 2^N$. The origin of the image is at the top-left corner and the coordinate of the bottom-right pixel is $(2^N - 1, 2^N - 1)$. The centroid $(\overline{X}, \overline{Y})$ is defined by

$$\overline{X} = \Sigma x_i / A,$$
$$\overline{Y} = \Sigma y_i / A,$$

where (x_i, y_i), $1 \leqslant i \leqslant k$, is the coordinate of each black pixel and A $(= k)$ is the area of the given image.

Given a CCS, the recursive decoding procedure must be performed first to get the coordinate of each black pixel. The following is the process for computing the sum of x-coordinates and the sum of y-coordinates for the black pixels represented by the CCS. Initially, the input parameters *sumx* and *sumy* are set to be zeros. Except to the actions of summing x_i's and summing y_i's, the **recursive procedure** *sum_xy*$(x, y, sumx, sumy)$ is similar to the **recursive procedure** *decode_CCS*(x, y) in Section 3. The sub-sum of all the x_i's and all the y_i's of the subimage G will be easily passed to the first phase to obtain the centroid value of the whole image. Since the computational bound required in computing the centroid is the same as that in computing the area, we omit the related complexity analysis.

```
recursive procedure sum_xy(x, y, sumx, sumy);
/* The direction[ ] is a constant variable.*/
/* The visited[ , ] is a global variable.*/
/* The i, dx, dy, and bit are local variables. */
  begin
    for i = 1 to 8 do
     begin
       (dx, dy) = direction[i];
       if ((x + dx, y + dy) is within the subimage G)
         and (visited[x + dx, y + dy] = false) then
           set visited[x + dx, y + dy] to be true;
           bit = next_CCS(); /* Get next bit in CCS. */
           if bit is '1' then
             sumx = sumx + x + dx; /* Sum up the decoded x-coordinate value.
       */
             sumy = sumy + y + dy; /* Sum up the decoded y-coordinate value.
       */
             sum_xy(x + dx, y + dy, sumx, sumy); /* Recursively decode the
       neighbor. */
           end if;
       end if;
     end for;
  end procedure;
```

5. Experimental results

In this section, under three binary images, first some experiments are included to demonstrate the compression improvements and the encoding-time improvement of our proposed two-phase SDS over the conventional four one-phase SDSs, such as the LQ, the DF-expression, the S-tree, and the bincode representation. Next the computational improvements of our proposed two-phase SDS for computing the area and the centroid, respectively, are demonstrated. As shown in Fig. 8a–c, respectively, the three used binary images are the butterfly, the

(a) Butterfly. (b) Floodmap. (c) Cup.

Fig. 8. Three testing images.

floodmap, and the cup. Each image is of size 256×256 and requires 65,536 bits. The experiments are performed on the IBM compatible personal computer Pentium III microprocessor with 667 MHz and 128 MB RAM. The operation system is MS-Windows 2000 and the program developing environment is Borland C++ Builder 5.0.

5.1. Compression and encoding-time improvements

The input image is first encoded using the methods of LQ, DF-expression, S-tree, and the bincode representation. Their compression (encoding-time) costs in terms of the number of bits (10^{-6} s, i.e., microseconds for Table 2) are shown in the third column of Table 1 (Table 2). Next, applying the CCS to the gray leaves from the first phase at a specific level, a better compression ratio is obtained. The fourth column denotes the number of bits (microseconds for Table 2) needed in our proposed two-phase SDS. Four different kinds of the subimages' sizes corresponding to the four one-phase SDSs, respectively, are shown in the fifth column. The compression (encoding-time) improvement ratios are shown in the last column and they reveal that our improved two-phase representation over the conventional SDSs have 66.14, 19.49, 12.33, and 48.83% compression (12.69, 4.66, 2.49, and 18.71% encoding-time) improvement ratios when compared to the LQ, DF-expression, S-tree representation, and the bincodes, respectively. The encoding-time improvements of our proposed two-phase SDS over the conventional one-phase SDSs listed in Table 2.

Table 1
Compression improvement

Method	One-phase M_1 bits	Two-phase M_2 bits	CCS coded subimage size	Improvement ratio $(M_1 - M_2)/M_1$ (%)
Butterfly				
LQ	30,399	11,143	8×8	63.34
DF	9330	7242	2×2	22.37
S-tree	7898	6854	1×2	13.21
Bincode	42,208	20,302	2×4	51.90
Floodmap				
LQ	20,388	6751	8×8	66.88
DF	4794	4066	2×2	15.18
S-tree	4148	3746	1×2	9.69
Bincode	19,648	11,573	2×4	41.09
Cup				
LQ	30,342	9648	8×8	68.20
DF	9386	7422	2×2	20.92
S-tree	8483	7287	1×2	14.09
Bincode	40,768	18,952	2×4	53.51

Table 2
Encoding-time improvement

Method	One-phase T_1	Two-phase T_2	CCS coded subimage size	Improvement ratio $(T_1 - T_2)/T_1$ (%)
Butterfly				
LQ	55,019	48,289	8×8	12.23
DF	46,157	43,903	2×2	4.88
S-tree	68,078	66,708	1×2	2.01
Bincode	81,708	63,572	2×4	22.19
Floodmap				
LQ	39,828	37,875	8×8	4.90
DF	37,524	36,102	2×2	3.78
S-tree	54,329	53,517	1×2	1.49
Bincode	59,336	52,405	2×4	11.68
Cup				
LQ	53,647	42,401	8×8	20.96
DF	46,707	44,214	2×2	5.33
S-tree	69,080	66,336	1×2	3.97
Bincode	82,629	64,212	2×4	22.28

Table 3
Execution-time improvement for computing area

Method	One-phase T_1	Two-phase T_2	CCS coded subimage size	Improvement ratio $(T_1 - T_2)/T_1$ (%)
Butterfly				
LQ	9084	1452	8×8	84.01
DF	9384	6279	2×2	33.08
S-tree	11,747	10,415	1×2	11.33
Bincode	10,475	5418	2×4	48.27
Floodmap				
LQ	4677	1061	8×8	77.31
DF	4837	3735	2×2	22.78
S-tree	5939	5578	1×2	6.07
Bincode	5478	3465	2×4	36.74
Cup				
LQ	9143	1462	8×8	84.00
DF	9413	6479	2×2	31.16
S-tree	12,748	11,126	1×2	12.72
Bincode	11,236	5809	2×4	48.30

5.2. Execution-time improvements for geometrical operations

Table 3 (Table 4) lists the computation improvements of our proposed two-phase method over the conventional one-phase methods for computing the area

Table 4
Execution-time improvement for computing centroid

Method	One-phase T_1	Two-phase T_2	CCS coded subimage size	Improvement ratio $(T_1 - T_2)/T_1$ (%)
Butterfly				
LQ	9564	4076	8×8	57.38
DF	9083	6418	2×2	29.34
S-tree	11,937	10,656	1×2	10.73
Bincode	10,334	6259	2×4	39.43
Floodmap				
LQ	4847	2624	8×8	45.86
DF	4677	3617	2×2	22.66
S-tree	6029	5501	1×2	8.75
Bincode	5418	3895	2×4	28.11
Cup				
LQ	9614	3475	8×8	63.85
DF	9133	6611	2×2	27.61
S-tree	12,869	11,201	1×2	12.96
Bincode	11,076	6670	2×4	39.77

(centroid). Using the four one-phase methods, the third column shows the execution-time in terms of microseconds for computing the area (centroid). Using our two-phase method, the fourth column shows the execution-time in terms of microseconds for computing the area (centroid). The execution-time improvement ratios for computing the area (centroid) are shown in the last column of Table 3 (Table 4) and they reveal that our improved two-phase method over the conventional one-phase methods have 81.77, 29.00, 10.04, and 44.43% (55.69, 26.53, 10.81, and 35.77%) improvement ratios when compared to the LQ-based method, DF-expression-based method, S-tree representation-based method, and the bincodes-based method, respectively.

6. Conclusion

We have presented the two-phase SDS to reduce the memory requirement required in the conventional tree-based SDSs. The related promising time complexity analyses are also provided. Experimental results show that our improved two-phase representation over the conventional SDSs have 66.14, 19.49, 12.33, and 48.83% memory improvement ratios (12.69, 4.66, 2.49, and 18.71% coding time improvement ratios) when compared to the LQ, DF-expression, S-tree representation, and the bincodes, respectively. We also show that our proposed two-phase representation has a better computational performance when running geometric operations, such as computing the area and the centroid, on the proposed two-phase representation directly. For computing the area (centroid), experimental results reveal that our improved two-phase SDS over the conventional one-phase SDSs have 81.77, 29.00,

10.04, and 44.43% (55.69, 26.53, 10.81, and 35.77%) improvement ratios when compared to the LQ, DF-expression, S-tree representation, and the bincodes, respectively.

Recently, Chung and Wu (2000) presented an improved S-Tree method, called the STC method, for compressing gray images. The STC method improves the execution time of Distasi et al.'s method (Distasi et al., 1997) in the ratio less than 1/2 while preserving the same image quality and bits rates. The STC method can be viewed as an extension of the conventional SDS from binary images to gray images. Although under the same bpp, the JPEG (Wallace, 1991) has a better PSNR when compared to the STC method. However, since the regular geometrical relationship among those partitioned blocks are reserved in the linear tree table of the STC method, the STC method has fruitful applications in image manipulations (Chung et al., 2002). How to plug our proposed two-phase representation of this paper into the STC method in order to improve the related performance for compressing gray images is the future research issue.

Acknowledgments

The authors appreciate the anonymous referees and S.Y. Tseng for their valuable comments that helped to improve the presentation and quality of the paper. This research was supported by the National Science Council of R.O.C. under contracts NSC89-2218-E011-018 and NSC90-2213-E011-056.

References

Chai, B.B., Vass, J., Zhung, X., 1999. Significance-linked connected component analysis for wavelet image coding. IEEE Trans. Image Process. 8 (6), 774–784.

Chung, K.L., Wu, J.G., 2000. Improved image compression using S-tree and shading approach. IEEE Trans. Commun. 48 (5), 748–751.

Chung, K.L., Yan, W.M., Liao, Z.H., 2002. Fast computation of moments on compressed grey images using block representation. Real-Time Imaging 8 (2), 137–144.

Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C., 2001. Introduction to Algorithms, second ed The MIT Press, New York.

Distasi, R., Nappi, M., Vitulano, S., 1997. Image compression by B-tree triangular coding. IEEE Trans. Commun. 45 (9), 1095–1100.

Gargantini, I., 1982. An effective way to represent quadtrees. Commun. ACM 25 (12), 905–910.

Gonzalez, R.C., Woods, R.E., 2002. Digital Image Processing, second ed Prentice-Hall, New Jersey.

Jonge, W.D., Scheuermann, P., Schijf, A., 1994. S^+-trees: an structure for the representation of large pictures. Comput. Vision Image Understand. 59, 265–280.

Kawaguchi, E., Endo, T., 1980. On a method of binary picture representation and its application to data compression. IEEE Trans. Pattern Anal. Mach. Intell. 2 (1), 27–35.

Ouksel, M.A., Yaagoub, A., 1992. The interpolation-based bintree and encoding of binary images. CVGIP: Graph. Models Image Process. 54 (1), 75–81.

Samet, H., 1990a. Applications of Spatial Data Structures. Addison-Wesley, New York.

Samet, H., 1990b. The Design and Analysis of Spatial Data Structures. Addison-Wesley, New York.

Serra, J.P., 1982. Image Analysis and Mathematical Morphology. Acdemic Press, New York.

Shaffer, C.A., Juvvadi, R., Health, L.S., 1993. Generalized comparison of quadtree and bintree storage requirements. Image Vision Comput. 11 (7), 402–412.
Wallace, G.K., 1991. The JPEG still picture compression standard. Commun. ACM 34 (4), 30–44.