Computers and Mathematics with Applications 58 (2009) 484–497

Contents lists available at ScienceDirect

Computers and Mathematics with Applications

journal homepage: www.elsevier.com/locate/camwa

Speed up of the edge-based inverse halftoning algorithm using a finite state machine model approach

Kuo-Liang Chung [a,*], Ping-Zen Chen [a], Ying-Lin Pan [b]

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

[b] Department of Information Management, National Taiwan University of Science and Technology, No. 43, Section 4, Keelung Road, Taipei 10672, Taiwan, ROC

ARTICLE INFO

Article history:
Received 7 March 2008
Received in revised form 6 February 2009
Accepted 17 April 2009

Keywords:
Edges
Finite state machine model
Halftone image
Inverse halftoning
Lookup table
Speedup

ABSTRACT

The recently published edge- and lookup table-based inverse halftoning (ELIH) algorithm has shown its quality and superiority when compared with the previous lookup table-based IH algorithm. This paper presents a new finite state machine model (FSMM)-based search method to speed up the existing ELIH algorithm significantly while preserving the same image quality as in the ELIH algorithm. From the observation that the sliding window for the ELIH algorithm moves from left to right one position; there are therefore one output column and one input column which are introduced at each step and thus a simple finite state machine can track the transitions from the current window movement to the next. This is faster than a full search in the lookup table. Under thirty typical testing images adopted from Meşe's website, experimental results demonstrated that the proposed FSMM-based ELIH algorithm has an improvement in execution time of 20% to 80%, with a typical improvement of 50%, when compared to the ELIH algorithm.

© 2009 Published by Elsevier Ltd

1. Introduction

Halftoning is a process which transforms gray images into halftone images [1]. It has been widely used in publishing applications, such as newspapers, books, magazines, etc. Halftone images are hard to manipulate. Many image operations, such as scaling, compression, and enhancement can cause severe image degradation [2]. To enable these operations, gray images need to be reconstructed from the halftone images through inverse halftoning (IH).

Since there is no way to reconstruct a perfect gray image from the given halftone image, many efficient IH algorithms [2–6] have been developed to improve the quality of the reconstructed image. Based on the lookup table (LUT) approach, three efficient IH algorithms [7–9] have been presented. Among these developed LUT-based IH (LIH) algorithms, Chang et al. [7] presented a hybrid IH algorithm which combines the LUT approach and a filtering technique. Meanwhile, Meşe and Vaidyanathan [8] independently presented the same LUT approach for IH. Recently, Chung and Wu [10] presented an edge-based LIH (ELIH) algorithm which has better image quality although the ELIH algorithm needs more memory requirement and search-time load when compared to the LIH algorithm.

Based on the recently published ELIH algorithm mentioned above, this paper presents a new finite state machine model (FSMM)-based search method to speed up the ELIH algorithm significantly while preserving the same image quality as in the ELIH algorithm. Our proposed FSMM-based ELIH (FELIH) method first analyzes the correlation between two edge patterns in the thirty nine edge patterns used in the existing ELIH and then, a two-level FSMM is constructed. According to the row-major scanning order on halftone images in the IH process, the constructed two-level FSMM can help the current scanned

* Corresponding author.
E-mail address: k.l.chung@mail.ntust.edu.tw (K.-L. Chung).

0898-1221/$ – see front matter © 2009 Published by Elsevier Ltd
doi:10.1016/j.camwa.2009.04.018

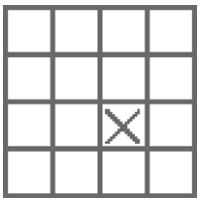


Fig. 1. A 4×4 template T.

binary pattern and the associated edge pattern to find the corresponding constructed gray value in the edge-based LUT (ELUT) efficiently. In fact, the proposed FSMM-based search scheme also can be used to speed up the construction of ELUT. When compared to the full search technique used in the previous ELIH, the proposed FELIH can narrow the search space efficiently for each search in average and it leads to a good computation-saving effect for reconstructing the gray image from the input halftone.

The remainder of this paper is organized as follows. In Section 2, the currently published ELIH algorithm is surveyed. In Section 3, the proposed FELIH algorithm is presented. In Section 4 the execution-time improvement of the proposed algorithm is demonstrated. In Section 5, some concluding remarks are addressed.

2. The past work: ELIH

In this section, we first survey the recently published ELIH algorithm and then point out the time-consuming problem existing in the ELIH.

Fig. 1 illustrates a 4×4 template T with symbol X which is used as a sliding window to build up the ELUT. Given a set of thirty testing halftone images from Meşés website, first we run one of the existing IH algorithms [7–9] on the testing halftone images to obtain the base gray images, i.e. the inverse halftoned image. These testing halftone images are also called the training halftone images since they will be used to construct the ELUT for the ELIH algorithm. Then, we run the Canny edge detector [11] on the base gray images to obtain the edge maps.

From the edge map, each 4×4 subedge map covered by the 4×4 template T is examined and then is classified into one of the four regular edge types or the irregular edge type. As shown in Fig. 2, the four regular edge types contain the horizontal edge type (see the twelve horizontal edge patterns as shown in Fig. 2(a)), the vertical edge type (see the twelve vertical edge patterns as shown in Fig. 2(b)), the diagonal edge type (see the six diagonal edge patterns as shown in Fig. 2(c)), and the corner edge type (see the four corner edge patterns as shown in Fig. 2(d)). The irregular edge type is partitioned into five groups, group-1 with index 34 containing the edge pattern without any edge pixels; group-2 denoted by index 35, group-3 by index 36, group-4 by index 37, and group-5 by index 38 containing edge patterns with 1–4, 5–8, 9–12, and 13–16 random edge pixels, respectively.

After introducing how to classify the edge types from the training edge maps, we now introduce the address mapping scheme in the ELIH algorithm. Based on the current training halftone image and the raster scan order, the input 4×4 binary pattern S^h denotes the 4×4 halftone covered by the template T. The sixteen binary values of S^h are denoted by S_0^h, S_1^h, …, S_{14}^h, and S_{15}^h. As the first key, S^h is encoded by

$$I = \sum_{k=0}^{15} 2^k S_k^h. \tag{1}$$

Using the 4×4 subedge map corresponding to the input 4×4 halftone as the second key, the index J can be obtained by searching the matched edge pattern in four regular edge types and the irregular edge type. For each input 4×4 halftone, the index pair (I, J) is used in the address mapping. Then, the lookup table ELUT is updated by the following assignments:

$$\begin{aligned} N[I, J] &= N[I, J] + 1 \\ ELUT[I, J] &= ELUT[I, J] + S_{10}^g \end{aligned} \tag{2}$$

where the array $N[I, J]$ is used as a counter and S_{10}^g denotes the T-mapped gray value of S^g pointed by symbol X in Fig. 1; here S^g denotes the corresponding 4×4 base gray subimage. Using the template T as the sliding window, we update the counter $N[*, *]$ (see Eq. (2)) and the array $ELUT[*, *]$ iteratively once moving T one position from left to right. Finally, we have

$$ELUT[I, J] = \frac{ELUT[I, J]}{N[I, J]}. \tag{3}$$

After introducing the construction of the 2-D array $ELUT[*, *]$ used in the previous ELIH algorithm, we observe that although the value of index I can be calculated by several arithmetic operations (see Eq. (1)), it still needs to compare the current 4×4 subedge map to each edge pattern from the regular edge pattern with index 0 to the irregular edge pattern with index 38

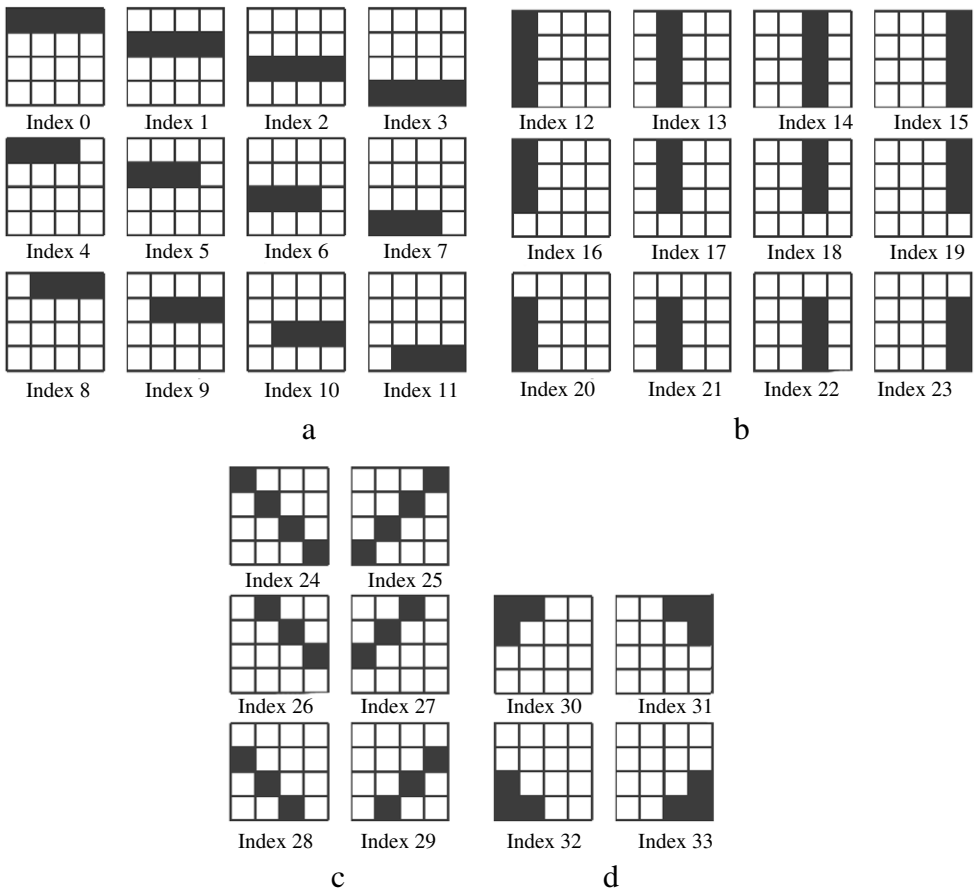


Fig. 2. Four regular edge types in the ELIH algorithm. (a) Horizontal edge type: twelve horizontal edge patterns. (b) Vertical edge type: twelve vertical edge patterns. (c) Diagonal edge type: six diagonal edge patterns. (d) Corner edge type: four corner edge patterns.

in a full search manner, and then the index of the matched edge pattern is assigned to be the value of J. Experimental data illustrates that the ratio of the execution-time required in determining the first key I over that in determining the second key J is 0.872%, so in our study, we ignore the time requirement for the first key and we only focus on the second key.

Unfortunately, in order to obtain the second key J, the full search approach used in the ELIH needs thirty eight comparisons in the worst case to compare the 4×4 subedge map with the thirty eight edge patterns. For each comparison, it takes sixteen bit-comparison operations. For obtaining the value of J for each scanned 4×4 subedge map, in total it takes 608 ($=16 \times 38$) bit-comparison operations. On the other hand, for each 4×4 subedge map, it takes a large amount of time to find the matched edge pattern among the thirty nine edge patterns. Besides this time-consuming problem, the reconstruction of a gray image via the ELUT has the same time-consuming search problem.

In the LIH algorithm, the memory required for the LUT is 64 K bytes. The ELUT has 39 edge types for each binary pattern; it, therefore, needs 2496 K (39×64 K) bytes. If we simply store a 64 K-byte lookup table mapping the 16-bit value of every 4×4 edge map to its J-value; although the J-value of each dot position could be computed in constant time, it needs $2^{16} \times 64$ K bytes in total and is not so practical.

3. The proposed FSMM-based inverse halftoning algorithm

Instead of using the full search approach mentioned above, this section presents the proposed FSMM-based search scheme to reduce the search time in the construction of ELUT and the reconstruction of a gray image.

As shown in Fig. 3(a), suppose the reference 4×4 subedge map is denoted by the symbol S^r; the current 4×4 subedge map is denoted by S^c, and the new input 4×1 column vector is denoted by V^c. The size of the overlapping subedge map between S^r and S^c is 4×3. Given the current 4×4 subedge map, our main task is to find the matched edge pattern in the set of thirty nine edge patterns. Since we have known the J-value of the reference 4×4 subedge map S^r, the purpose of the constructed FSMM can help us to find the J-value of the current 4×4 subedge map S^c efficiently. Our main idea is that upon knowing the J-value of S^r, e.g. 15, if the column vector V^c is an empty vector, $(0, 0, 0, 0)^t$, it is easy to know the J-value of S^c, i.e. 14. Throughout this paper, a 4×1 column vector V^c is denoted by $V^c = (b_1, b_2, b_3, b_4)^t$, where $b_i = 0(=1)$ denotes the

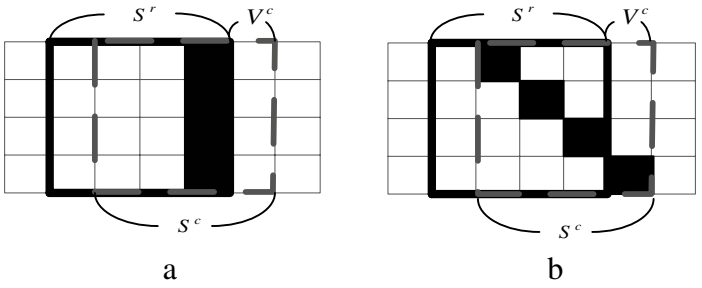


Fig. 3. Two examples for reference subedge map S^r and current subedge map S^c. (a) Reference subedge map S^r with J-value 15 and current subedge map S^c with J-value 14. (b) Reference subedge map S^r with J-value 26 and current subedge map S^c with J-value 24.

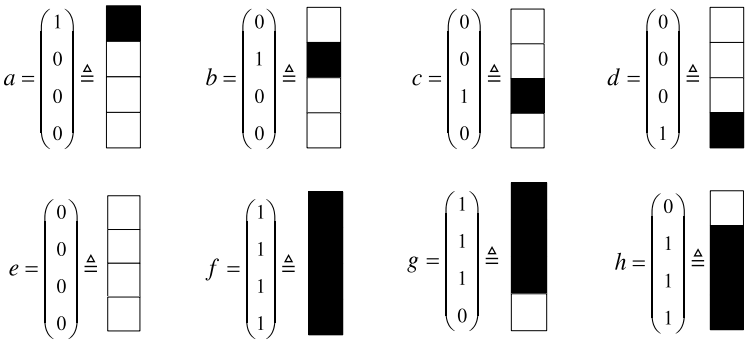


Fig. 4. The set of eight feasible column vectors, V_F.

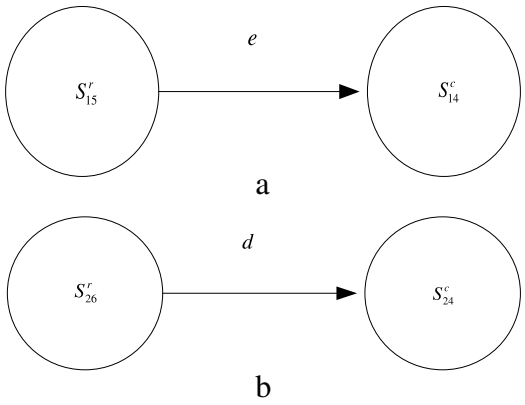


Fig. 5. Two transitions for Fig. 3. (a) Transition for Fig. 3(a). (b) Transition for Fig. 3(b).

black (white) pixel of V^c. By the same argument, upon knowing the J-value of S^r, e.g. 26 (see Fig. 3(b)), if the column vector V^c is equal to $(0, 0, 0, 1)^t$, the J-value of S^c is 24. Given an input column vector V^c, the transition from [13] the J-value of the reference 4×4 subedge map S^r to the J-value of the current 4×4 subedge map S^r can be realized by the constructed FSMM. In what follows, the construction of our proposed FSMM will be presented.

For exposition, we now define the frequently used eight column vectors and they are useful to speed up the search time for determining the J-value of the current 4×4 subedge map. As shown in Fig. 4, the eight 4×1 column vectors are defined by $a = (1, 0, 0, 0)^t$, $b = (0, 1, 0, 0)^t$, $c = (0, 0, 1, 0)^t$, $d = (0, 0, 0, 1)^t$, $e = (0, 0, 0, 0)^t$, $f = (1, 1, 1, 1)^t$, $g = (1, 1, 1, 0)^t$, and $h = (0, 1, 1, 1)^t$ where the symbol 't' denotes the transpose operation. For convenience, these eight column vectors are called feasible column vectors and they are represented by the set $V_F = \{a, b, c, d, e, f, g, h\}$. Since the total number of all possible column vectors is 16 ($=2^4$), the ratio of the number of feasible column vectors over that of all possible column vectors is $1/2(=8/16)$. Let V denote the set of all possible column vectors, then the set $V_{IF} = V \setminus V_F$ denotes the set of eight infeasible column vectors.

After describing the definitions of V_F and V_{IF}, Fig. 5(a), (b) depict the corresponding two transitions where S^r_i (S^c_j) denotes the state of reference (*current*) subedge map with J-value $i(j)$. For the example in Fig. 5(a), the state S^r_{15} denotes the reference subedge map with J-value 15; after reading the column vector $e = (0, 0, 0, 0)^t$, the transition tells us that the new state

Table 1
The transition function and output function of HFSM.

$\acute{S}_H$	I_H									
	a	b	c	d	e	$V \setminus \{a, e\}$	$V \setminus \{b, e\}$	$V \setminus \{c, e\}$	$V \setminus \{d, e\}$	V
S_0	$S_0, 0$				$S_4, 0$	$S_{\text{stop}}, 1$				
S_1		$S_1, 0$			$S_5, 0$		$S_{\text{stop}}, 1$			
S_2			$S_2, 0$		$S_6, 0$			$S_{\text{stop}}, 1$		
S_3				$S_3, 0$	$S_7, 0$				$S_{\text{stop}}, 1$	
S_4										$S_{\text{stop}}, 1$
S_5										$S_{\text{stop}}, 1$
S_6										$S_{\text{stop}}, 1$
S_7										$S_{\text{stop}}, 1$
S_8	$S_0, 0$				$S_4, 0$	$S_{\text{stop}}, 1$				
S_9		$S_1, 0$			$S_5, 0$		$S_{\text{stop}}, 1$			
S_{10}			$S_2, 0$		$S_6, 0$			$S_{\text{stop}}, 1$		
S_{11}				$S_3, 0$	$S_7, 0$				$S_{\text{stop}}, 1$	

for the current subedge map will be S^c_{14} and can be determined in constant time. Following the transition notation used in the FSMM, the two transitions in Fig. 5(a), (b) can be given by $f_s{:}S^r_{15} \times e \to S^c_{14}$ and $f_s{:}S^r_{26} \times d \to S^c_{24}$, respectively. Without confusion, for convenience, the above two transitions are denoted by $f_s : S_{15} \times e \to S_{14}$ and $f_s{:}S_{26} \times d \to S_{24}$. From finite state machine notations, S_{15}, S_{14}, S_{26}, and S_{24} are four states where the state S_i denotes the subedge map with J-value i. In fact, among the four regular edge types mentioned in Fig. 2, only the 4×4 reference subedge map, which belongs to the horizonal edge type, the vertical edge type, or the diagonal edge type, associated with an input column vector could determine the J-value of the 4×4 current subedge map fast. According to the example of the above two transitions in Fig. 5, our FSMM consists of three different FSMMs, namely the horizontal FSM (HFSM), the vertical FSM (VFSM), and the diagonal FSM (DFSM). The three different kinds of FSMMs will be defined formally in what follows.

Before defining the HFSM, let us first discuss how to group the suitable edge types as the state set used in the HFSM. From Fig. 2(a), the twelve states S_0, S_1, ..., and S_{11} corresponding to the twelve horizontal edge patterns can be grouped into a state set in the HFSM. In addition, let S_{stop} denote the final state in the HFSM and this special state tells us that the next 4×4 subedge map does not belong to the horizontal edge type. That is, when the state comes to S_{stop}, the HFSM will be terminated temporarily and the full search is alarmed and performed for the next 4×4 subedge map. From Fig. 2(a), we observe that when the reference subedge map belongs to the horizontal edge type, the current subedge map still belongs to the same edge type if the input 4×1 column vector contains only one black pixel or no black pixels. For example, suppose that 4×4 subedge map is S_0 and the new input column vector is e, then it indicates that the next 4×4 subedge map S_4 belongs to horizontal edge type; suppose the 4×4 subedge map is S_0 and the new input column vector is a, then it indicates that the next 4×4 subedge map S_0 also belongs to a horizontal edge type. Generally speaking, assume the reference subedge map belongs to the horizontal edge type, and when the input column vector is equal to a, b, c, d, or e (see Fig. 4), the next state may be one of S_0, S_1, ..., and S_{11}; otherwise, the next state is S_{stop} since the next subedge map does not belong to the horizontal edge type. The definition of HFSM is given below.

Definition 1. Finite State Machine for Horizonal Edge Type (HFSM): The HFSM M_H is represented by $M_H = (S_H, I_H, O_H, f_{H_s}, f_{H_o})$ where the state set S_H is denoted by $S_H = \{S_0, S_1, \ldots, S_{11}\} \bigcup \{S_{\text{stop}}\} = \acute{S}_H \bigcup \{S_{\text{stop}}\} = \{S_0, S_1, \ldots, S_{11}, S_{\text{stop}}\}$; the input column vector set I_H is denoted by $I_H = V = \{(b_1, b_2, b_3, b_4)^{\text{t}} | b_i = 1 \text{ or } b_i = 0, 1 \le i \le 4\}$; the output set O_H is denoted by $O_H = \{0, 1\}$; the transition function f_{H_s} and the output function f_{H_o} for M_H are defined by Table 1 where $f_{H_s} : \acute{S}_H \times I_H \to \acute{S}_H \bigcup \{S_{\text{stop}}\}$ and $f_{H_o} : \acute{S}_H \times I_H \to O_H$. The state diagram of the HFSM is shown in Fig. 6 which is equal to Table 1.

By Table 1 or Fig. 6, given the current state and the input column vector, the next state and the output can be determined. For example, given the state S_8 and the input column vector a, by Table 1, the next state and the output are S_0 and 0, respectively. As shown in Fig. 7, a small example is given to explain how the HFSM works. The first 4×4 subedge map is S_8, i.e. the initial state is S_8, and the new input 4×1 column vector is a. According to the state diagram of HFSM, from S_8 and a, we can determine the next state to be S_0 and the output to be 0. Next, based on the state S_0 and the input vector e, the HFSM tells us that the next state is S_4 and the output is 0. As shown in Fig. 7, the final input vector is b, the transition function transfers the state S_4 to S_{stop} and the output function gives the output 1. At this moment, the state S_{stop} stops the functionality of the HFSM and then we call the full search procedure to compute the J-value of the current 4×4 subedge map.

After explaining how the HFSM works, we continue discussing how to group the suitable edge types as the state set used in the FSM for the vertical edge type (VFSM). From Fig. 2(b), the twelve states S_{12}, S_{13}, ..., and S_{23} corresponding to the twelve vertical edge patterns can be grouped into the state set in the VFSM. Similarly, the state S_{stop} defined in the HFSM denotes the final state in the VFSM, but this special state tells us that the next 4×4 subedge map does not belong to the vertical edge type. From Fig. 2(b), we observe that when the reference subedge map belongs to the vertical edge type, the current subedge map still belongs to the same edge type if more than three continuous elements of the input 4×1 column vector are black pixels. For example, suppose the reference 4×4 subedge map is S_{12} and the new input column vector f has four continuous black pixels, we can find that the next 4×4 subedge map S_{15} also belongs to vertical edge type. Generally

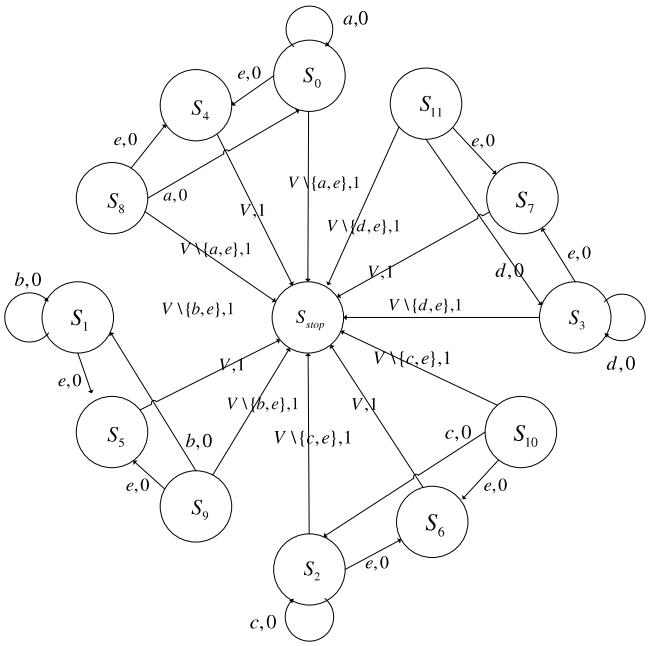


Fig. 6. The state diagram of HFSM.

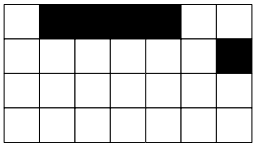


Fig. 7. A small example to explain how the HFSM works.

speaking, assume the reference subedge map belongs to the vertical edge type, if the input column vector is equal to f, g, h, or e (see Fig. 4), the next state may be one of $S_{12}, S_{13}, \ldots$, and S_{23}; otherwise, the next state is S_{stop}.

Let us further examine the transition relations between the state S_{34}, which denotes a white 4×4 subedge map, and any state of S_{12}, S_{16}, and S_{20}. We thus have the following transition functions and output functions: $f_s{:}S_{12} \times e \to S_{34}$; $f_o{:}S_{12} \times e \to 0$, $f_s : S_{16} \times e \to S_{34}$; $f_o{:}S_{16} \times e \to 0$, $f_s{:}S_{20} \times e \to S_{34}$; $f_o{:}S_{20} \times e \to 0$, $f_s{:}S_{34} \times f \to S_{15}$; $f_o{:}S_{34} \times f \to 0$, $f_s{:}S_{34} \times g \to S_{19}$; $f_o{:}S_{34} \times g \to 0$, $f_s{:}S_{34} \times h \to S_{23}$; $f_o{:}S_{34} \times h \to 0$, $f_s{:}S_{34} \times e \to S_{34}$; $f_o{:}S_{34} \times e \to 0$, $f_s{:}S_{34} \times (V \setminus \{e, f, g, h\}) \to S_{\text{stop}}$; $f_o{:}S_{34} \times (V \setminus \{e, f, g, h\}) \to 1$. From the thirty obtained edge maps as shown in Fig. 18, it is observed that for an edge map, white 4×4 subedge maps occupy a large portion of that edge map. Besides the transitions between any two vertical edge patterns in Fig. 2(b), the above eight transition functions and eight output functions can also speed up the J-value determination efficiently. The definition of VFSM is given below.

Definition 2. Finite state machine for vertical edge type (VFSM): The VFSM M_V is represented by $M_V = (S_V, I_V, O_V, f_{V_s}, f_{V_o})$ where the state set S_V is denoted by $S_V = \{S_{12}, S_{13}, \ldots, S_{23}, S_{34}\} \bigcup \{S_{\text{stop}}\} = \acute{S}_V \bigcup \{S_{\text{stop}}\} = \{S_{12}, S_{13}, \ldots, S_{23}, S_{34}, S_{\text{stop}}\}$; the set of input column vectors I_V is denoted by $I_V = V = \{(b_1, b_2, b_3, b_4)^{\text{t}} | b_i = 1 \text{ or } b_i = 0, 1 \le i \le 4\}$; the output set O_V is denoted by $O_V = \{0, 1\}$; the transition function f_{V_s} and the output function f_{V_o} for M_V are defined by Table 2 where $f_{V_s}{:}\acute{S}_V \times I_V \to \acute{S}_V \bigcup \{S_{\text{stop}}\}$ and $f_{V_o}{:}\acute{S}_V \times I_V \to O_V$. The state diagram of the VFSM is shown in Fig. 8.

As shown in Fig. 9, a small example is still given to explain how the VFSM works. Given the first 4×4 subedge map S_{12} and the new input column vector f, it yields the next state S_{15} and the output 0. Based on the state S_{15} and the input vector e, the VFSM yields the next state S_{14} and the output 0.

Finally, the six states $S_{24}, S_{25}, \ldots$, and S_{29} corresponding to the six diagonal edge patterns (see Fig. 2(c)) are grouped into the state set in the DFSM. The state S_{stop} is the same as in the VFSM. From Fig. 2(c), we observe that when the reference subedge map belongs to the diagonal edge type, the current subedge map still belongs to the same edge type if only the first or last element of the input column vector is the black pixel or the four elements of the input column vector are white pixels. For example, suppose that the reference 4×4 subedge map is S_{29} and new input column vector is a, we find that the next 4×4 subedge map S_{25} also belongs to diagonal edge type; suppose the reference 4×4 subedge map is S_{24} the new input column vector is e, we can find that the next 4×4 subedge map S_{28} also belongs to diagonal edge type. In summary, assume the reference subedge map belongs to the diagonal edge type, when the input column vector is equal to a, d, or e

Table 2
The transition function and output function of VFSM.

$\acute{S}_V$	I_H								
	f	g	h	e	$V \setminus \{f, e\}$	$V \setminus \{g, e\}$	$V \setminus \{h, e\}$	$V \setminus \{e\}$	$V \setminus \{e, f, g, h\}$
S_{12}	$S_{15}, 0$			$S_{34}, 0$	$S_{\text{stop}}, 1$				
S_{13}				$S_{12}, 0$				$S_{\text{stop}}, 1$	
S_{14}				$S_{13}, 0$				$S_{\text{stop}}, 1$	
S_{15}				$S_{14}, 0$				$S_{\text{stop}}, 1$	
S_{16}		$S_{19}, 0$		$S_{34}, 0$		$S_{\text{stop}}, 1$			
S_{17}				$S_{16}, 0$				$S_{\text{stop}}, 1$	
S_{18}				$S_{17}, 0$				$S_{\text{stop}}, 1$	
S_{19}				$S_{18}, 0$				$S_{\text{stop}}, 1$	
S_{20}			$S_{23}, 0$	$S_{34}, 0$			$S_{\text{stop}}, 1$		
S_{21}				$S_{20}, 0$				$S_{\text{stop}}, 1$	
S_{22}				$S_{21}, 0$				$S_{\text{stop}}, 1$	
S_{23}				$S_{22}, 0$				$S_{\text{stop}}, 1$	
S_{34}	$S_{15}, 0$	$S_{19}, 0$	$S_{23}, 0$	$S_{34}, 0$					$S_{\text{stop}}, 1$

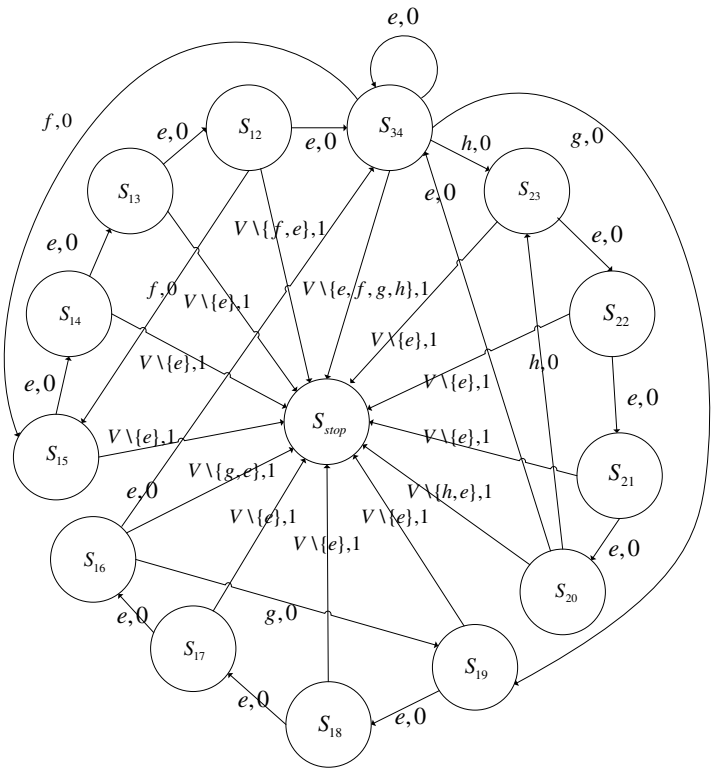


Fig. 8. The state diagram of VFSM.

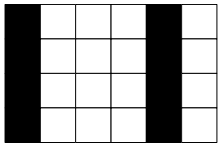


Fig. 9. A small example to explain how the VFSM works.

(see Fig. 4), the next state may be the one of $S_{24}, S_{25}, \ldots,$ and S_{29}; otherwise, the next state is S_{stop}. The definition of DFSM is given below.

Definition 3. *Finite state machine for diagonal edge type (DFSM)*: The DFSM M_D is represented by $M_D = (S_D, I_D, O_D, f_{D_s}, f_{D_o})$ where the state set S_D is denoted by $S_D = \{S_{24}, S_{25}, \ldots, S_{29}\} \bigcup \{S_{\text{stop}}\} = \acute{S}_D \bigcup \{S_{\text{stop}}\} = \{S_{24}, S_{25}, \ldots, S_{29}, S_{\text{stop}}\}$; the set of input column vectors I_D is denoted by $I_D = V = \{(b_1, b_2, b_3, b_4)^t | b_i = 1 \text{ or } b_i = 0, 1 \leq i \leq 4\}$; the output set O_D

Table 3
The transition function and output function of DFSM.

$\acute{S}_D$	I_H						
	a	d	e	$V \setminus \{a\}$	$V \setminus \{d\}$	$V \setminus \{e\}$	V
S_{24}			$S_{28}, 0$			$S_{\text{stop}}, 1$	
S_{25}			$S_{27}, 0$			$S_{\text{stop}}, 1$	
S_{26}		$S_{24}, 0$			$S_{\text{stop}}, 1$		
S_{27}							$S_{\text{stop}}, 1$
S_{28}							$S_{\text{stop}}, 1$
S_{29}	$S_{25}, 0$			$S_{\text{stop}}, 1$			

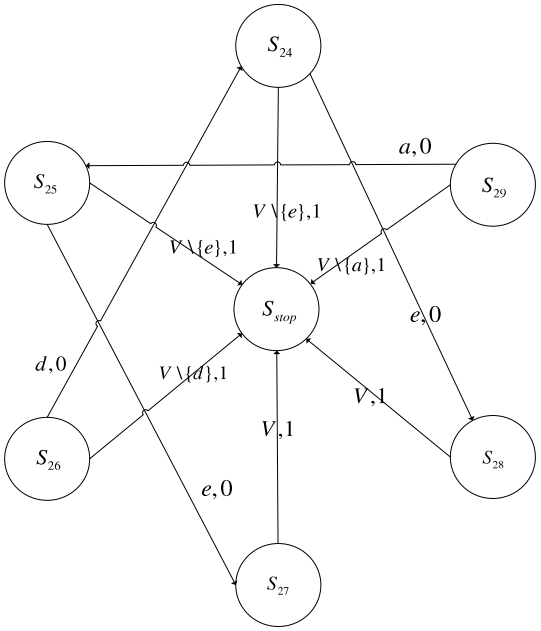


Fig. 10. The state diagram of DFSM.

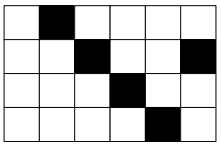


Fig. 11. A small example to explain how the DFSM works.

is denoted by $O_D = \{0, 1\}$; the transition function f_{D_s} and the output function f_{D_o} for M_D are defined by Table 3 where $f_{D_s} : \acute{S}_D \times I_D \to \acute{S}_D \bigcup \{S_{\text{stop}}\}$ and $f_{D_o} : \acute{S}_D \times I_D \to O_D$. The state diagram of the DFSM is shown in Fig. 10.

As shown in Fig. 11, the first 4×4 subedge map is S_{26} and the new input column vector is d. From S_{26} and d, the next state is S_{24} and the output is 0. The final input vector is an infeasible vector which belongs to $V \setminus \{e\}$, so the transition function transfers the state S_{24} to S_{stop} and the output function gives the output 1. We thus stop the functionality of the DFSM and then call the full search procedure to compute the J-value of the current 4×4 subedge map.

Based on Definitions 1–3, our FSMM consists of the HFSM, the VFSM, and the DFSM. As shown in Fig. 12, we combine the above three examples (see Fig. 7, Fig. 9, and Fig. 11) to explain how our proposed whole FSMM works. Since each time the 4×4 template T is moved one position from left to right in Fig. 12, there are sixteen different 4×4 subedge maps as shown in Fig. 13 that can be scanned. Fig. 14 shows our proposed whole FSMM and Table 4 illustrates the detailed simulation for Fig. 12.

The main contribution of this paper is that in our proposed FSMM, suppose the J-value of reference 4×4 subedge map is within the interval [0..29], when the new input 4×1 column vector belongs to the feasible column vectors V_F (see Fig. 4), the proposed FSMM can determine the J-value of next 4×4 subedge map in constant time, and it leads to a good computation-saving effect. In next section, some experimental results are used to demonstrate the good execution-time improvement of our proposed FELIH.

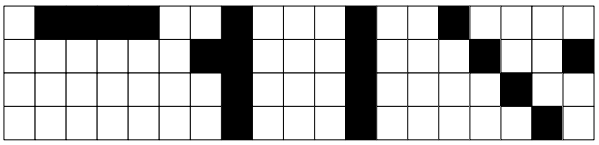


Fig. 12. An example to explain how our proposed whole FSMM works.

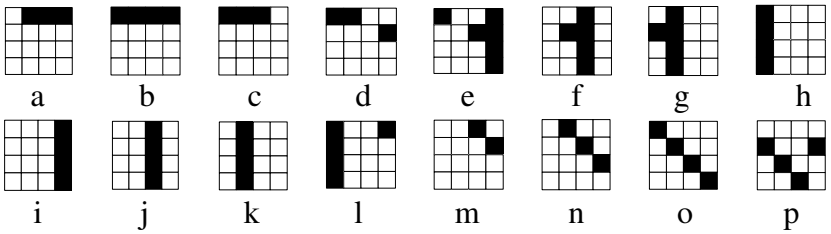


Fig. 13. Sixteen different 4×4 subedge maps for Fig. 12.

Table 4
The simulation of our proposed FSMM for Fig. 12.

Step	Operation	Current state	Input column vector	Next state	Output	J-value	4×4 template
1	Full search					8	Fig. 13(a)
2	HFSM	S_8	a	S_0	0	0	Fig. 13(b)
3	HFSM	S_0	e	S_4	0	4	Fig. 13(c)
4	HFSM	S_4	b				
5	Full search					35	Fig. 13(d)
6	Read next column vector						
7	Full search					36	Fig. 13(e)
8	Read next column vector						
9	Full search					36	Fig. 13(f)
10	Read next column vector						
11	Full search					36	Fig. 13(g)
12	Read next column vector						
13	Full search					12	Fig. 13(h)
14	VFSM	S_{12}	f	S_{15}	0	15	Fig. 13(i)
15	VFSM	S_{15}	e	S_{14}	0	14	Fig. 13(j)
16	VFSM	S_{14}	e	S_{13}	0	13	Fig. 13(k)
17	VFSM	S_{13}	a	S_{stop}	1		
18	Full search					36	Fig. 13(l)
19	Read next column vector						
20	Full search					35	Fig. 13(m)
21	Read next column vector						
22	Full search					26	Fig. 13(n)
23	DFSM	S_{26}	d	S_{24}	0	24	Fig. 13(o)
24	DFSM	S_{24}	b	S_{stop}	1		
25	Full search					35	Fig. 13(p)

4. Experimental results

In this section, some experimental results are demonstrated to justify the execution-time improvement of our proposed FELIH when compared to the previous ELIH. All the experiments are implemented by using a Pentium 4 personal computer with 2.6 GHz and 512 MB RAM. The operating system used is MS-Windows XP and the program developing environment is Borland C++ builder 6.0.

In our experiments, the thirty test 768×512 gray images are adopted from Meşe's website [12]. For simplicity, we only choose three typical images, say the fifth gray image for the worst execution-time improvement, the ninth gray image for middle execution-time improvement, and the twenty-sixth gray image for the best execution-time improvement. The three typical test images are illustrated in Fig. 15. Fig. 16 shows the halftone images generated by running error diffusion process with Floyd–Steinberg kernel on Fig. 15. After applying the LIH in [7,8] to these halftone images, the base gray images are shown in Fig. 17 and the corresponding edge maps are illustrated in Fig. 18 by running Canny edge detector on these base gray images. Fig. 19 shows the positions where the dot symbol '·' denotes the kernel-mapped pixel of the current 4×4 edge map whose J-value can be determined in constant time by the proposed FSMM. For the three edge maps in Fig. 19, in average, the ratio of the number of dots in one edge map over the size of the edge map, i.e. 768×512, is about 60%. The execution-time comparison and execution-time improvement ratios for the thirty testing images are shown in Table 5 where execution-time is measured in terms of seconds. From Table 5, experimental results indicate that for each

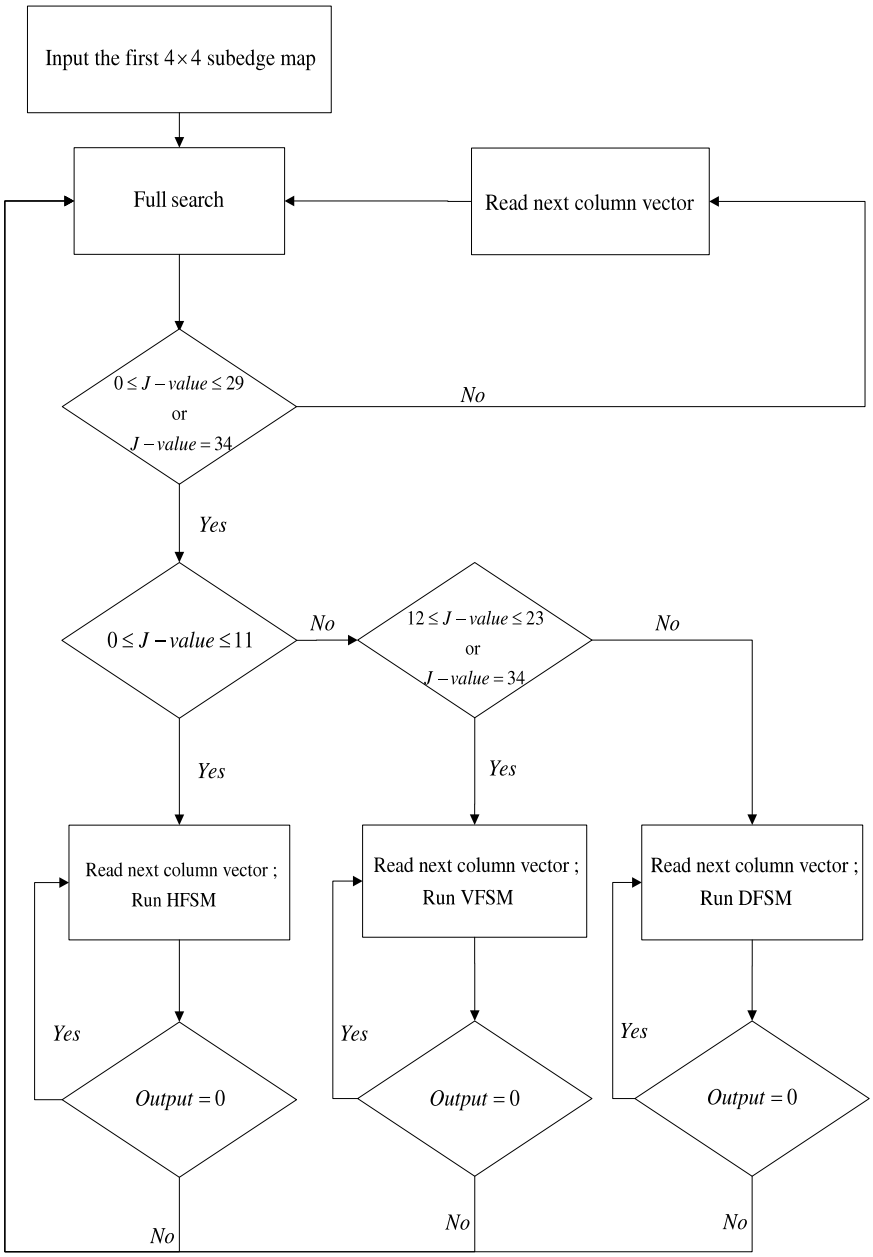


Fig. 14. Our proposed whole FSMM.

edge map, the average execution-time for determining the J-values by using our proposed FELIH (the ELIH) is 1.277 (2.742) seconds and our proposed FELIH is faster than the previous ELIH; the average execution-time improvement ratio is about 50% ($\cong \frac{2.742-1.277}{2.742} \times 100\%$). Note that if the VFSM does not include the consideration of S_{34} for the white 4×4 subedge map, the average execution-time improvement ratio is only 22%.

5. Conclusion

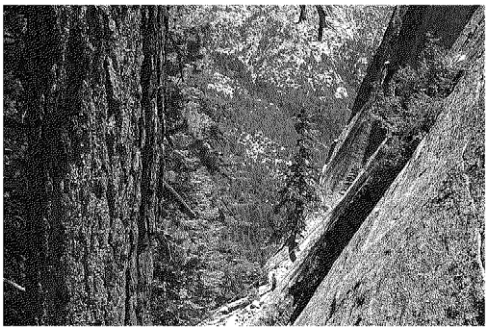
Based on the finite state machine model (FSMM) approach, we have presented a faster FSMM-based IH, called the FELIH, to speed up the currently published ELIH algorithm significantly. Our proposed two-level FSMM consists of three kinds of finite state machine (FSM), the HFSM, the VFSM, and the DFSM, and each of them can help us to determine the J-value of the current 4×4 subedge map efficiently. Under thirty different testing images, experimental results have demonstrated that our proposed FELIH has about 50% execution-time improvement ratio on average when compared to the previous ELIH algorithm.

(a) The fifth gray image.

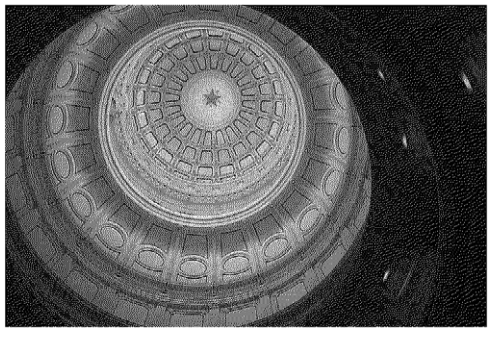
(b) The ninth gray image.

(c) The twenty-sixth gray image.

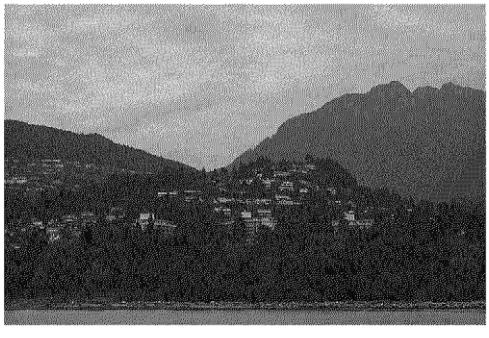
Fig. 15. Three typical testing gray images.



(a) The fifth halftone image.



(b) The ninth halftone image.



(c) The twenty-sixth halftone image.

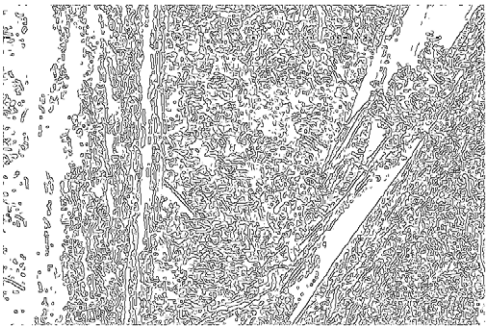
Fig. 16. The halftone images generated by running error diffusion process with Floyd–Steinberg kernel on Fig. 15.

(a) The fifth base gray image.

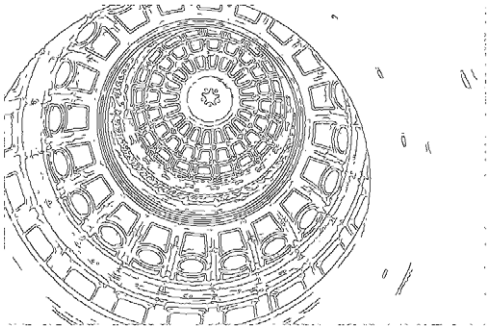
(b) The ninth base gray image.

(c) The twenty-sixth base gray image.

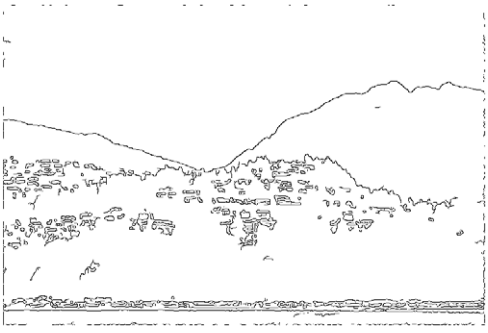
Fig. 17. Three typical base gray images.



(a) The fifth edge map.

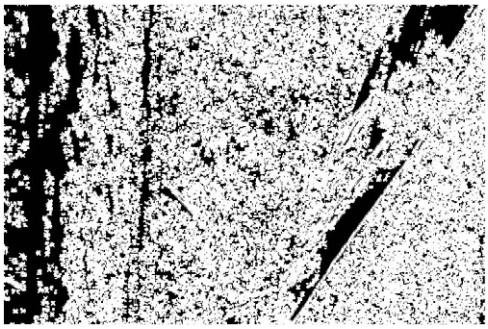


(b) The ninth edge map.

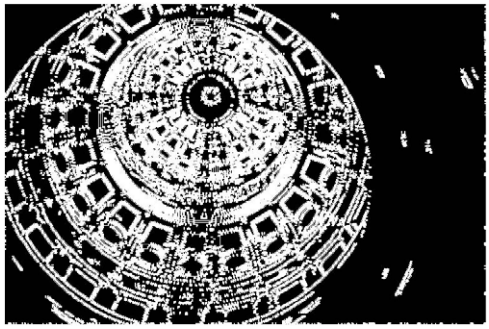


(c) The twenty-sixth edge map.

Fig. 18. Three typical edge maps obtained by Canny edge detector.



(a) Black dot positions for the fifth image.



(b) Black dot positions for the ninth image.

(c) Black dot positions for the twenty-sixth image.

Fig. 19. The black dot positions that those J-values can be determined in constant time by using our proposed FSMM-based approach.

Table 5
Execution-time improvement ratios between the ELIH and the FELIH.

Image	T_{ELIH} (s)	T_{FELIH} (s)
Image 1	2.9	1.8
Image 2	2.9	1.7
Image 3	2.8	1.3
Image 4	2.6	1.0
Image 5	3.1	2.3
Image 6	2.7	1.1
Image 7	2.9	1.6
Image 8	2.6	0.9
Image 9	2.7	1.3
Image 10	2.4	0.6
Image 11	2.7	1.4
Image 12	2.8	1.3
Image 13	2.5	0.7
Image 14	2.9	1.6
Image 15	2.7	1.1
Image 16	2.6	0.9
Image 17	2.8	1.4
Image 18	2.7	1.1
Image 19	2.9	1.6
Image 20	2.6	0.9
Image 21	2.8	1.6
Image 22	2.6	1.0
Image 23	2.4	0.5
Image 24	2.8	1.3
Image 25	3.1	2.3
Image 26	2.4	0.4
Image 27	2.7	1.2
Image 28	2.8	1.3
Image 29	2.6	0.9
Image 30	3.1	2.2
Average	2.7	1.3
$\frac{T_{\mathrm{ELIH}}-T_{\mathrm{FELIH}}}{T_{\mathrm{ELIH}}}$	52%	

Acknowledgement

The first author was supported by the National Science Council of R. O. C. under contracts NSC96-2221-E011-153 and NSC97-2218-E011-128.

References

[1] R.A. Ulichney, Digial Halftoning, MIT Press, Cambridge, MA, 1987.
[2] T.D. Kite, N. Damera-Venkata, B.L. Evans, A.C. Bovik, A fast, high-quality inverse halftoning algorithm for error diffused halftones, IEEE Trans. Image Process. 9 (2000) 1583–1592.
[3] Z.C. Lai, J.Y. Yen, Inverse error-diffusion using classified vector quantization, IEEE Trans. Image Process. 7 (1998) 1753–1758.
[4] M.Y. Shen, C.-C.J. Kuo, A robust nonlinear filtering approach to inverse halftoning, J. Vis. Commun. Image Represent. 12 (2001) 84–95.
[5] R.L. Stevenson, Inverse halftoning via MAP estimation, IEEE Trans. Image Process. 6 (1997) 574–583.
[6] Z. Xiong, M.T. Orchard, K. Ramchandran, Inverse halftoning using wavelets, IEEE Trans. Image Process. 8 (1999) 1479–1482.
[7] P.C. Chang, C.S. Yu, T.H. Lee, Hybrid LMS-MMS inverse halftoning technique, IEEE Trans. Image Process. 10 (2001) 95–103.
[8] M. Meşe, P.P. Vaidyanathan, Look up table (LUT) method for inverse halftoning, IEEE Trans. Image Process. 10 (2001) 1566–1578.
[9] M. Meşe, P.P. Vaidyanathan, Tree-Structured method for LUT inverse halftoning and for image halftoning, IEEE Trans. Image Process. 11(2002) 644–655.
[10] K.L. Chung, S.T. Wu, Inverse halftoning algorithm using edge-based lookup table approach, IEEE Trans. Image Process. 14 (2005) 1583–1589.
[11] J. Canny, A computational approach to edge detection, IEEE Trans. Pattern Anal. Mach. Intell. 8 (1986) 679–698.
[12] Online. Available: http://www.systems.caltech.edu/mese/halftone/.
[13] H.R. Lewis, C.H. Papadimitriou, Elements of the Theory of Computation, Prentice-Hall, Inc., 1998, (Chapter 2).