

## Short Paper

---

# A Fast Finite-State Algorithm for Generating RGB Palettes of Color Quantized Images\*

YU-LEN HUANG AND RUEY-FENG CHANG<sup>+</sup>

*Department of Computer Science and Information Engineering*

*Tunghai University*

*Taichung, 407 Taiwan*

*E-mail: ylhuang@mail.thu.edu.tw*

<sup>+</sup>*Department of Computer Science and Information Engineering*

*National Chung Cheng University*

*Chiayi, 621 Taiwan*

*E-mail: rfchang@cs.ccu.edu.tw*

On the WWW, the transmission time for videos and images impacts the performance of a web site. In order to reduce the bandwidth that is used to transmit images over the Internet, most image formats adopt only a limited number of colors used simultaneously to display color images on a video monitor. Hence, generating a good color palette for a color digitized image is an important task for Internet applications. In general, the Linde-Buzo-Gray (LBG) algorithm can be used to cluster a color digitized image in which each pixel is considered as a 3-dimension vector in an RGB color space for generating a color palette. The codebook generated by the LBG algorithm can be considered as the color palette for the color image. In order to obtain a good color palette, the LBG algorithm needs a large amount of computation time. In this paper, we propose a color finite-state LBG (CFSLBG) algorithm that reduces the computation time by exploiting the correlations of palette entries between the current and previous iterations. Instead of searching the whole color palette, the CFSLBG algorithm searches only a small number of colors that are very close to the training vector. Thus, the computation time for color quantization is reduced. The proposed approach generates RGB palettes efficiently with little sacrifice of quantized image quality. This paper describes the implementation of this work and simulation results.

**Keywords:** palettes generating, color quantization, LBG algorithm, finite-state algorithm, VQ

## 1. INTRODUCTION

Video monitors typically use the three primary color components, red, green, and blue, to specify the color of each pixel in a color image. Each primary component usually provides 8 bits for specifying the color of each pixel in a full-color digitized image.

---

Received January 21, 2002; revised January 21 & April 24, 2003; accepted May 8, 2003.

Communicated by Ming-Syan Chen.

\* This work was supported by the National Science Council, Taiwan, R.O.C., under Grand NSC 91-2213-E-029-021.

There has been steady growth in the Internet resources utilization for the last decade. The World Wide Web (WWW), which combines multimedia and networking techniques, is widely used to access multimedia information on the Internet. Hence, the WWW has become the most widely used part of the Internet. The transmission time required for videos and images impacts the performance of a WWW site. In order to reduce the network bandwidth required to transmit the images over the Internet, most of the color image formats adopt only a limited number of colors to simultaneously display full-color images on display devices. Generally speaking, 256 colors are enough to display a color image. The set of these quantized 256 colors is called a color map or palette. The monitor uses the palette to display a color image. Each entry in a color palette contains a 24-bit value, which specifies the three color components. Thus, in order to achieve better network performance, a full-color digitized image is usually quantized using a 256-color palette.

Color quantization algorithms can be grouped into splitting algorithms and clustering-based algorithms. Splitting algorithms iteratively split the color space of the original image into color subspaces [1, 2]. Then, the representative color of each subspace becomes the quantized color. Generally, splitting algorithms are fast. However, they cannot always obtain the optimal solution because the splitting operations cannot be resumed. On the other hand, clustering-based algorithms extract quantized colors by applying various clustering algorithms. The algorithms may generate an optimal color palette, but these approaches are very time consuming and complicated [3-8]. In this paper, we propose an effortless and straightforward clustering algorithm, which is fast and excellent for generating palette for a color image.

Vector quantization (VQ) has been shown to be an efficient method of image coding [9-12]. The input vectors are individually quantized to the closest codeword in the codebook. The codebook is generated by using some clustering algorithms from a set of training images. Image compression is achieved by transmitting the codeword indices for the information of the encoded images. Image decompression is done by utilizing the indices as addresses to the corresponding codewords in the decoder's codebook. Generating a good codebook is the key step in VQ. The iterative clustering algorithm proposed by Linde, Buzo, and Gray (LBG) is usually used in VQ [3]. A number of methods for generating color palettes using the VQ codebook design techniques were proposed in [13-15]. In view of color image coding, each pixel can be considered as a 3-dimension vector in the RGB color space. These vectors are the input vectors of the color VQ (CVQ) scheme. The LBG algorithm is most popular method used to select a color palette with a limited number of colors from a full-color digitized image.

In each iteration of the LBG algorithm, it searches the whole color palette in order to find the corresponding palette entry for each training vector. That is, the LBG algorithm requires a large amount of computation for color quantization. This paper proposes a novel color finite-state LBG (CFSLBG) algorithm that reduces the computation time required to select palettes from color images. Instead of searching the whole color palette, the proposed algorithm searches only a small part of the palette to find the corresponding palette entry for each training vector. In the CFSLBG algorithm, the number of palette entries that need to be searched for a training vector in each iteration is always much smaller than the size of the whole palette. For this reason, the duration of each iteration is greatly reduced. The computation time of the CFSLBG algorithm is much smaller than

that of the LBG algorithm.

The rest of this paper is organized as follows. We discuss the VQ technique and LBG algorithm in section 2. Section 3 presents the structure of the proposed CFSLBG algorithm. Experimental results are given in section 4. Finally, conclusions are drawn in section 5.

## 2. LBG ALGORITHM FOR GENERATING COLOR PALETTES

A vector quantization is defined as a mapping from a  $k$ -dimension Euclidean space  $R^k$  to a finite subset of  $R^k$ . This finite set  $C = \{\hat{x}_i : i = 1, \dots, N\}$  is called a codebook, where  $N$  is the size of the codebook. Each vector  $\hat{x}_i = (\hat{x}_{i0}, \dots, \hat{x}_{i(k-1)})$  in codebook  $C$  is called a codeword. The codebook used by the VQ encoder and decoder is generated by using an iterative clustering algorithm, such as the LBG algorithm.

In color digitized images, each pixel is considered as a 3-dimension vector in the RGB space. Each of these vectors contains three values for each of the primary components, red, green, and blue. In many applications, the codebook in VQ is considered as the color palette when a color digitized image is quantized. In general, a complete color VQ (CVQ) image coding includes three basic steps. First, an index  $i$ , which points to the closest color vector  $\hat{x}$  in the color palette, is assigned to each input vector  $x = (x_R, x_G, x_B)$  by the CVQ encoder. Then, the index  $i$  is transmitted to the CVQ decoder. Finally, the CVQ decoder decompresses the image by using the transmitted indices to find the corresponding color from the color palette. Fig. 1 shows the basic structure of the CVQ scheme. The distortion between the input vector  $x$  and its corresponding palette entry  $\hat{x}$  is measured as the squared Euclidean distortion measure, i.e.,

$$d(x, \hat{x}) = \|x - \hat{x}\|^2 = (x_R - \hat{x}_R)^2 + (x_G - \hat{x}_G)^2 + (x_B - \hat{x}_B)^2. \quad (1)$$

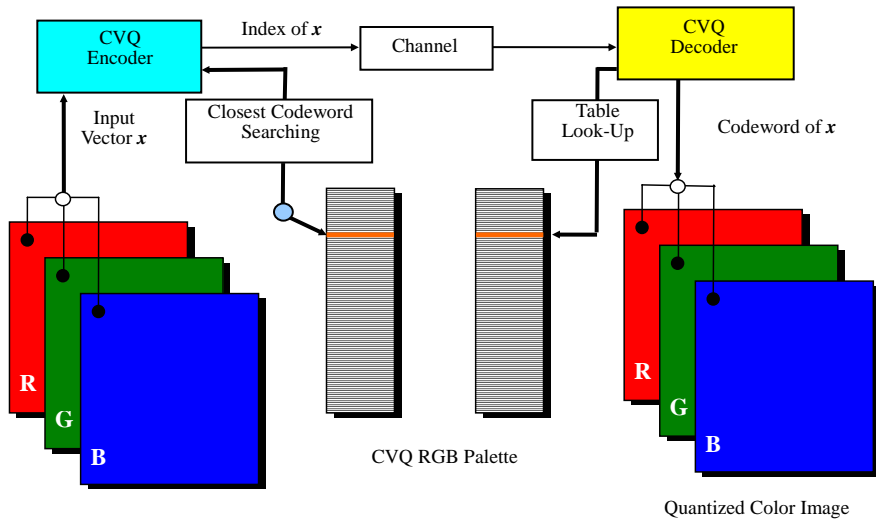


Fig. 1. The basic structure of the CVQ scheme.

In the LBG algorithm, color palette generation is an iterative process which converts the previous palette into the current palette from the training color image. The algorithm begins with a predefined initial color palette and repeats the mapping until the achieved improvement of the average distortion between the current and previous iterations is small enough. The performance of the LBG algorithm is dependent on the initial color palette. A number of techniques for designing the initial codebook were proposed in [16]. Then, an LBG algorithm iteration clusters each training vector to the closest entry in the previous color palette. Each palette entry is replaced by the centroid of the set of training vectors that are closer to this palette entry than to the others in the color palette. Let  $CS(\hat{\mathbf{x}})$  be the set of these training vectors. The centroid of the set  $CS(\hat{\mathbf{x}})$  is defined as

$$\text{centroid}(CS(\hat{\mathbf{x}})) = \frac{1}{|CS(\hat{\mathbf{x}})|} \sum_{\mathbf{x} \in CS(\hat{\mathbf{x}})} \mathbf{x}, \quad (2)$$

where  $|CS(\hat{\mathbf{x}})|$  denotes the number of training vectors in  $CS(\hat{\mathbf{x}})$ . The average distortion  $D_{AVG}$  of training vectors and their closest palette entries can be expressed as

$$D_{AVG} = \frac{1}{|T|} \sum_{\mathbf{x} \in T} d(\mathbf{x}, \hat{\mathbf{x}}), \quad (3)$$

where  $|T|$  is the number of training vectors in training set  $T$  and  $\hat{\mathbf{x}}$  is the closest palette entry for the training vector  $\mathbf{x}$ . The algorithm terminates when the difference of the average distortion between iteration  $i^{\text{th}}$  and  $(i-1)^{\text{th}}$  is smaller than a predefined distortion threshold  $\varepsilon$  ( $\varepsilon > 0$ ). Let  $P_i$  be the color palette generated in the  $i^{\text{th}}$  iteration, and let  $\hat{\mathbf{x}}^i$  be a palette entry in  $P_i$ . The LBG algorithm for color image coding is concisely given as follows:

- Step 1:** Design an initial RGB color palette  $P_0$  and set  $i \leftarrow 1$ ,  $D_{AVG_0} \leftarrow \infty$ .
- Step 2:** For each training vector  $\mathbf{x}$ , find the closest palette entry  $\hat{\mathbf{x}}^{i-1}$  by searching the whole color palette  $P_{i-1}$ .
- Step 3:** Compute the average distortion  $D_{AVG_i}$ . If  $|D_{AVG_{i-1}} - D_{AVG_i}| / D_{AVG_i}$  is smaller than  $\varepsilon$ , then stop.
- Step 4:** For each entry  $\hat{\mathbf{x}}^{i-1}$  in  $P_{i-1}$ , generate a new entry  $\hat{\mathbf{x}}^i \leftarrow \text{centroid}(CS(\hat{\mathbf{x}}^{i-1}))$  and add  $\hat{\mathbf{x}}^i$  into  $P_i$ . Set  $i \leftarrow i + 1$  and go to step 2.

In step 2 of the LBG algorithm, each training vector requires searching throughout the entire color palette. Because of the inefficiency of full searching, the LBG algorithm requires a large amount of computation to obtain a good color palette from the training image. We have observed that the current palette entry for a training vector  $\mathbf{x}$  is usually very close to the palette entry for  $\mathbf{x}$  in the previous iteration. Therefore, we propose a modified LBG algorithm in this paper that searches only a small number of palette entries for  $\mathbf{x}$ . Because it does not search the whole color palette, the computation time required by our algorithm for color palette generation is much reduced.

### 3. COLOR FSLBG ALGORITHM

Although the VQ scheme yields acceptable performance for image coding, the finite-state vector quantization (FSVQ) schemes [16-21] improve performance for an ordinary VQ. An FSVQ can be viewed as a finite collection of ordinary VQ's, each with its own codebook associated with a state, which is called the state codebook. The encoding state of the current input vector is decided by a state function  $F(\mathbf{x})$ . This coding state may be described by a state variable  $s \in S = \{s_i : i = 1, \dots, M\}$ , where  $M$  is the total number of states. The FSVQ is defined as a mapping from  $R^k \times S$  to a subset of a master codebook  $MC = \{\mathbf{x}_i : i = 1, \dots, N\}$ . For each state  $s_i$ , the FSVQ encoder selects  $N_f$  codewords by means of the state function from the master codebook  $MC$  as the state codebook  $SC_s$ . For each input vector  $\mathbf{x}$ , the encoder decides the current state  $s$  and then searches the state codebook  $SC_s$  to find its corresponding codeword. In the decompression phase, the decoder finds the same current state  $s$  and the corresponding codeword in the same state codebook  $SC_s$  by means of the transmitted index. The codebook size of the state codebook is much smaller than that of the master codebook. Hence, the searching time can be reduced, and the image quality can be maintained. A fast finite-state algorithm that reduces the computation time for vector quantizer design by exploiting FSVQ techniques was proposed in [22]. That paper shows that the fast finite-state algorithm can reserve the quality of encoded image. We notice that the finite-state technique was well suited to color image coding. Thus, this paper proposes a color finite-state LBG algorithm that modifies the training step of the ordinary LBG algorithm for generating a color palette.

Let the palette  $P_{i-1}$  be generated before the  $i^{\text{th}}$  iteration in the LBG algorithm, and let the previous codeword  $\hat{\mathbf{x}}^{i-1}$  be the closest color vector of  $P_{i-1}$  for a training vector  $\mathbf{x}$  in the  $(i-1)^{\text{th}}$  iteration. That is, the codeword  $\hat{\mathbf{x}}^{i-1}$  can be used as the state in the  $i^{\text{th}}$  iteration in our CFSLBG algorithm. Clearly, the state space in the  $i^{\text{th}}$  iteration of the CFSLBG algorithm is the palette  $P_{i-1}$ . For each state  $s$ , the state palette  $SP_s$  is the subset of whole palette  $P_i$ , and the size of  $SP_s$  is  $N_f$ . The  $N_f$  codewords in  $SP_s$  are the closest codewords to  $s$  in the whole palette  $P_{i-1}$ . The block diagram of generating the state palette in the CFSLBG algorithm is shown as Fig. 2. In the first iteration, there is no previous information for the CFSLBG algorithm. Thus, the first iteration of the CFSLBG algorithm is the same as that of the ordinary LBG algorithm, in which the full search algorithm is used to select a codeword for each training vector. At the following iteration, i.e.  $i \geq 2$ , the information in the previous iteration is used to determine the states of the training vectors. The CFSLBG algorithm is described in the following steps.

- Step 1:** Design an initial RGB color palette  $P_0$  and set  $D_{AVG_0} \leftarrow \infty$ .
- Step 2:** For each training vector  $\mathbf{x}$ , find the closest palette entry  $\hat{\mathbf{x}}^0$  by searching the whole color palette  $P_0$ . Compute the average distortion  $D_{AVG_1}$ .
- Step 3:** For each palette entry  $\hat{\mathbf{x}}^0$  in  $P_0$ , generate a new entry  $\hat{\mathbf{x}}^1 \leftarrow \text{centroid}(CS(\hat{\mathbf{x}}^0))$  and add  $\hat{\mathbf{x}}^1$  into  $P_1$ . Set  $i \leftarrow 2$ .
- Step 4:** Set the state space  $S = P_{i-1}$ .
- Step 5:** For each state  $s$  in  $S$ , find the set of the  $N_f$  closest codewords in the whole color palette  $P_{i-1}$  and define this set as the state palette  $SP_s$ . For each training vector  $\mathbf{x}$ , use the previous codeword  $\hat{\mathbf{x}}^{i-1}$  as the state  $s$ . Find the closest palette entry  $\hat{\mathbf{x}}^i$  by searching the state palette  $SP_s$ .

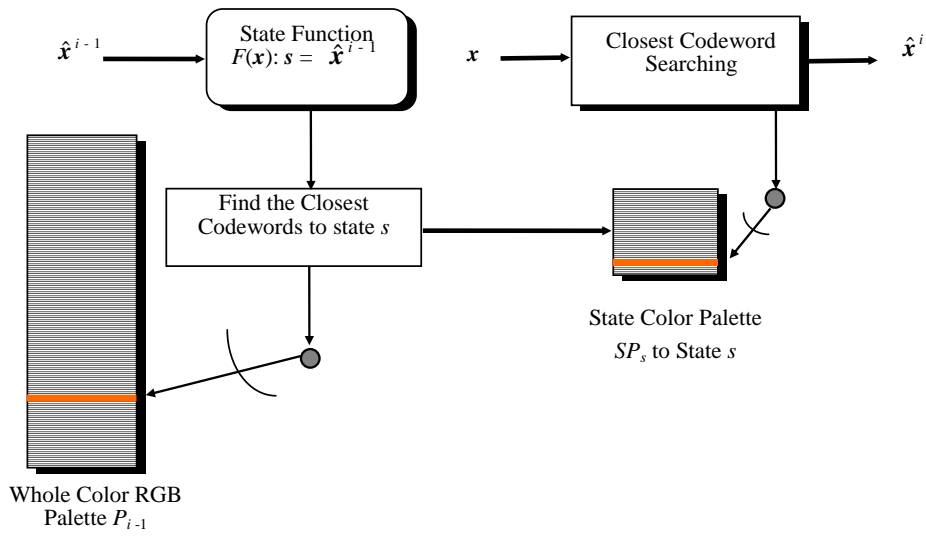


Fig. 2. Block diagram of generating the state palette in the iteration  $i$  ( $i \geq 2$ ) of the CFSLBG algorithm.

**Step 6:** Compute the average distortion  $D_{AVG_i}$ . If  $|D_{AVG_{i+1}} - D_{AVG_i}| / D_{AVG_i}$  is smaller than  $\varepsilon$ , then stop.

**Step 7:** For each entry  $\hat{x}^{i-1}$  in  $P_{i-1}$ , generate a new entry  $\hat{x}^i \leftarrow \text{centroid}(\text{CS}(\hat{x}^{i-1}))$  and add  $\hat{x}^i$  into  $P_i$ . Set  $i \leftarrow i + 1$  and go to step 4.

Note that the state palette size  $N_f$  is much smaller than the whole color palette size  $N$ . The  $N_f$  codewords for each state are found by an insertion sorting algorithm applied to the entire palette. The size of the state palette  $N_f$  is very small, such as 4, 8, 12, or 16. In this situation, the insertion sorting algorithm is efficient. The CFSLBG algorithm is much faster than the LBG algorithm. Moreover, the image quality is very close to that of the LBG algorithm, based on our experimental results described in section 4. Therefore, the CFSLBG algorithm is an effective method for generating palettes of color digitized images.

#### 4. SIMULATIONS AND RESULTS

Our CFSLBG algorithm and the LBG algorithm for generating color palettes in the RGB space were simulated on a SUN SPARC workstation IPX for several still  $512 \times 512$  RGB color images with 24 bits per pixel (bpp). To evaluate the performance of the color palette numerically, the RGB peak signal-to-noise ratio (PSNR) between the original color image and the quantized color image was calculated, where the RGB PSNR is defined as

$$\text{PSNR} = 10 \log_{10} \frac{255^2}{\text{MSE}_{\text{RGB}}} \text{ dB.} \quad (4)$$

Note that the mean squared error for an  $n \times n$  RGB color image is defined as

$$\text{MSE}_{\text{RGB}} = \frac{1}{3} \left( \left( \frac{1}{n} \right)^2 \sum_{i=1}^n \sum_{j=1}^n d(x_{ij}, \hat{x}_{ij}) \right), \quad (5)$$

where  $x_{ij}$  and  $\hat{x}_{ij}$  denote the original and quantized primary color intensity levels, respectively.

We compared the ordinary LBG algorithm and the CFSLBG algorithm in the simulations. Two distortion thresholds,  $\varepsilon_1 = 0.0001$  and  $\varepsilon_2 = 0.001$ , were used to generate a color palette that contained 256 colors from a color image. In the CFSLBG algorithm, the size of the whole palette is 256, and the size of state palette  $N_f$  is 4, 8, 12, or 16. Tables 1-2 show the number of iterations and execution times at a distortion threshold of  $\varepsilon = 0.0001$  for the test color images. The number of iterations and execution times at larger distortion threshold,  $\varepsilon = 0.001$ , for the test color images are listed in Tables 4-5. In our experiments, the execution time of the CFSLBG algorithm was only about 8% of the time required by the LBG algorithm. Tables 3 and 6 show the PSNR values for the color images with the different distortion threshold values. It can be seen that when the size of the state palette is larger than 8, the performance of the CFSLBG algorithm in terms of the RGB PSNR is very close to that of the LBG algorithm.

**Table 1. The numbers of iterations required by the CFSLBG and LBG algorithms ( $N = 256$  and  $\varepsilon = 0.0001$ ) for different images.**

Images	CFSLBG				LBG
	$N_f = 4$	$N_f = 8$	$N_f = 12$	$N_f = 16$	
Lena	46	48	53	50	41
Peppers	45	51	62	53	47
F-16	53	76	82	66	80
Sailboat	35	24	62	21	33
Tiffany	68	45	55	66	34

**Table 2. The execution times (in seconds) required by the CFSLBG and LBG algorithms ( $N = 256$  and  $\varepsilon = 0.0001$ ) for different images.**

Images	CFSLBG				LBG
	$N_f = 4$	$N_f = 8$	$N_f = 12$	$N_f = 16$	
Lena	62.3	73.4	95.0	96.9	1501.6
Peppers	61.3	112.0	156.8	168.5	1820.3
F-16	62.4	110.9	157.6	166.4	3140.6
Sailboat	58.6	66.1	145.8	86.3	1310.9
Tiffany	75.7	85.7	125.8	178.3	1404.4

**Table 3. The RGB PSNR of the CFSLBG and LBG algorithms ( $N = 256$  and  $\varepsilon = 0.0001$ ) for different images.**

Images	CFSLBG				LBG
	$N_f = 4$	$N_f = 8$	$N_f = 12$	$N_f = 16$	
Lena	35.006	35.286	35.303	35.321	35.336
Peppers	27.513	28.119	28.417	28.408	28.339
F-16	36.408	37.372	37.523	37.509	38.002
Sailboat	29.120	29.049	29.418	29.363	29.509
Tiffany	33.251	34.270	34.516	34.723	34.319

**Table 4. The numbers of iterations required by the CFSLBG and LBG algorithms ( $N = 256$  and  $\varepsilon = 0.001$ ) for different images.**

Images	CFSLBG				LBG
	$N_f = 4$	$N_f = 8$	$N_f = 12$	$N_f = 16$	
Lena	26	30	33	25	36
Peppers	37	46	56	49	41
F-16	43	56	63	59	46
Sailboat	22	17	30	15	25
Tiffany	64	46	49	52	30

**Table 5. The execution times (in seconds) required by the CFSLBG and LBG algorithms ( $N = 256$  and  $\varepsilon = 0.001$ ) for different images.**

Images	CFSLBG				LBG
	$N_f = 4$	$N_f = 8$	$N_f = 12$	$N_f = 16$	
Lena	50.5	66.7	87.1	87.4	1306.5
Peppers	58.9	91.3	137.4	154.9	1577.3
F-16	58.1	91.2	129.4	152.4	1812.1
Sailboat	50.7	57.3	90.0	72.4	1010.9
Tiffany	73.4	86.7	116.3	148.3	1241.6

**Table 6. The RGB PSNR of the CFSLBG and LBG algorithms ( $N = 256$  and  $\varepsilon = 0.001$ ) for different images.**

Images	CFSLBG				LBG
	$N_f = 4$	$N_f = 8$	$N_f = 12$	$N_f = 16$	
Lena	34.832	35.215	35.252	35.152	35.311
Peppers	27.515	27.996	28.344	28.373	28.324
F-16	36.247	37.241	37.341	37.488	37.587
Sailboat	29.113	29.034	29.367	29.358	29.491
Tiffany	33.236	34.271	34.508	34.692	34.295

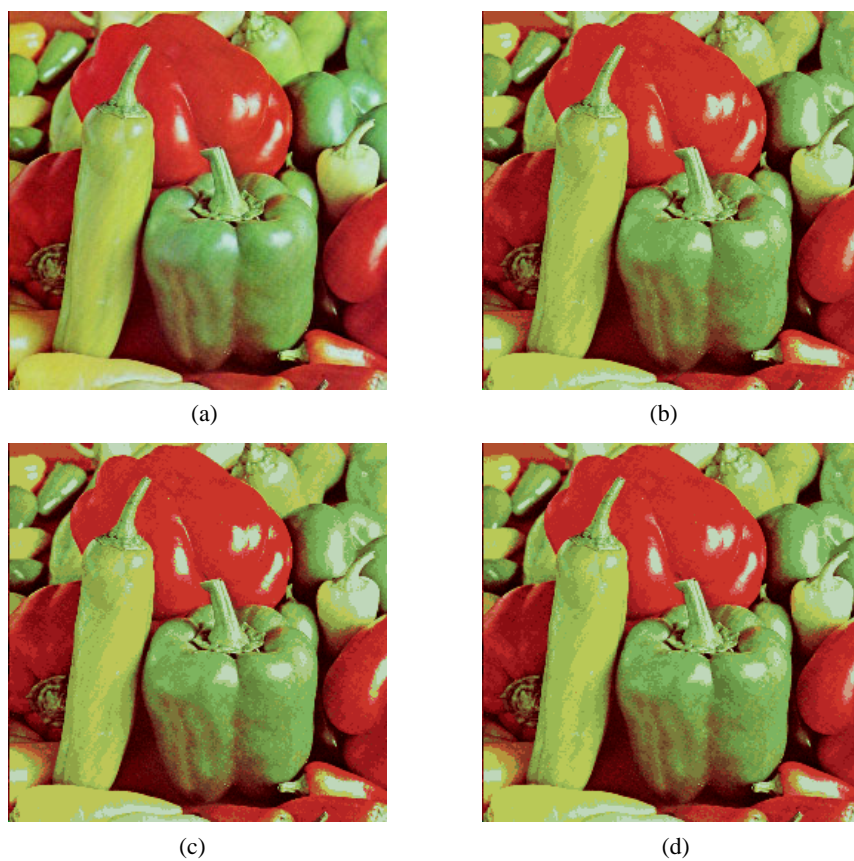


Fig. 3. Results for the image Peppers ( $N = 256$  and  $\varepsilon = 0.001$ ): (a) original image, (b) LBG algorithm, (c) CFSLBG algorithm ( $N_f = 4$ ), and (d) CFSLBG algorithm ( $N_f = 8$ ).



Fig. 4. Results for the image Lena ( $N = 256$  and  $\varepsilon = 0.001$ ): (a) original image, (b) LBG algorithm, (c) CFSLBG algorithm ( $N_f = 4$ ), and (d) CFSLBG algorithm ( $N_f = 8$ ).



Fig. 4. (Cont'd) Results for the image Lena ( $N = 256$  and  $\varepsilon = 0.001$ ): (a) original image, (b) LBG algorithm, (c) CFSLBG algorithm ( $N_f = 4$ ), and (d) CFSLBG algorithm ( $N_f = 8$ ).

The images Peppers and Lena shown in Figs. 3 (a) and 4 (a) are the original  $512 \times 512$  color images with 24 bpp. Figs. 3 (b)-(d) show the quantized results for the image Peppers when the distortion threshold  $\varepsilon$  was 0.001. Fig. 3 (b) shows the quantized image obtained using the color palette that was generated by the LBG algorithm. Figs. 3 (c)-(d) show the quantized results of the CFSLBG algorithm with  $N_f = 4$  and  $N_f = 8$ , respectively. Figs. 4 (b)-(d) show the quantized results for the image Lena when  $\varepsilon$  was 0.001. Fig. 4 (b) shows the quantized image obtained using the color palette that was generated by the LBG algorithm. Figs. 4 (c)-(d) show the quantized results of the CFSLBG algorithm with  $N_f = 4$  and  $N_f = 8$ , respectively.

## 5. CONCLUSIONS

In this paper, the CFSLBG algorithm for generating a RGB palette from a training color image has been proposed. The ordinary LBG algorithm ignores the correlation of corresponding palette entries between the current and previous iterations. In general, the current corresponding palette entry for a training vector is close to the previous corresponding palette entry. The CFSLBG algorithm searches only the palette entries that are close to the palette entry for the training vector in the previous iteration. However, the number of palette entries is much smaller than the size of the whole palette. This is the reason why our CFSLBG algorithm is faster than the ordinary LBG algorithm. In our experiment, the quality of the color quantized image obtained using the CFSLBG palette was very close to that of the image obtained using the LBG palette. Moreover, the computation time of the CFSLBG algorithm is only about 8% of the time required by the LBG algorithm. The experimental results demonstrate the excellent performance of the proposed approach in color quantization. We conclude that the CFSLBG algorithm is an effective method for designing RGB palettes of color digitized images.

## REFERENCES

1. G. Joy and Z. Xiang, "Center-cut for color image quantization," *The Visual Computer: International Journal of Computer Graphics*, Vol. 10, 1993, pp. 62-66.
2. C. Y. Yang and J. C. Lin, "RWM-cut for color image quantization," *Computer and Graphics*, Vol. 20, 1996, pp. 577-588.
3. Y. Linde, A. Buzo, and R. M. Gray, "An algorithm for vector quantizer design," *IEEE Transactions on Communications*, Vol. COM-28, 1980, pp. 84-95.
4. Y. W. Lim and S. U. Lee, "On the color image segmentation algorithm based on the thresholding and the fuzzy C-means techniques," *Pattern Recognition*, Vol. 23, 1990, pp. 935-952.
5. Z. Xiang, "Color image quantization by minimizing the maximum intercluster distance," *ACM Transactions on Graphics*, Vol. 16, 1997, pp. 260-276.
6. I. S. Hsieh and K. C. Fan, "An adaptive clustering algorithm for color quantization," *Pattern Recognition Letters*, Vol. 21, 2000, pp. 337-346.
7. E. Rendon, L. Salgado, J. M. Menendez, and N. Garcia, "Adaptive palette determination for color images based on Kohonen networks," in *Proceedings of the International Conference on Image Processing (ICIP)*, Vol. 1, 1997, pp. 830-833.
8. J. S. Kirk, D. J. Chang, and J. M. Zurada, "A self-organizing map with dynamic architecture for efficient color quantization," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, Vol. 3, 2001, pp. 2128-2132.
9. R. M. Gray, "Vector quantization," *IEEE ASSP Magazine*, 1984, pp. 4-29.
10. M. Goldberg, P. R. Boucher, and S. Shlien, "Image compression using adaptive vector quantization," *IEEE Transactions on Communications*, Vol. COM-34, 1986, pp. 180-187.
11. N. M. Nasrabadi and R. A. King, "Image coding using vector quantization: a review," *IEEE Transactions on Communications*, Vol. 36, 1988, pp. 957-971.
12. H. M. Hang and B. G. Haskell, "Interpolative vector quantization of color images," *IEEE Transactions on Communications*, Vol. 36, 1988, pp. 465-470.
13. W. H. Equitz, "A new vector quantization clustering algorithm," *IEEE Transactions on Acoustics, Speech, and Signal Processing (ASSP)*, Vol. 37, 1989, pp. 1568-1575.
14. M. T. Orchard and C. A. Bouman, "Color vector quantization of images," *IEEE Transactions on Signal Processing*, Vol. 39, 1991, pp. 2677-2690.
15. A. Zaccarin and B. Liu, "A novel approach for coding color vector quantized images," *IEEE Transactions on Signal Processing*, Vol. 2, 1993, pp. 442-453.
16. J. Foster, R. M. Gray, and M. O. Dunham, "Finite-state vector quantization of waveform coding," *IEEE Transaction on Information Theory*, Vol. IT-31, 1985, pp. 348-359.
17. R. Aravind and A. Gersho, "Low-rate image coding with finite-state vector quantization," in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 1986, pp. 137-140.
18. W. T. Chen, R. F. Chang, and J. S. Wang, "Image sequence coding using finite-state vector quantization," *IEEE Transaction on Circuits Systems for Video Technology*, Vol. 2, 1992, pp. 15-24.
19. M. O. Dunham and R. M. Gray, "An algorithm for the design of label-transition finite state vector quantizers," *IEEE Transactions on Communications*, Vol. COM-33, 1985,

- pp. 83-89.
20. R. F. Chang and W. T. Chen, "Image coding using variable-rate side-mach finite-state vector quantization," *IEEE Transactions on Image Processing*, Vol. 2, 1993, pp. 104-108.
  21. N. M. Nasrabadi and Y. Feng, "A dynamic finite-state vector quantization scheme," in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 1990, pp. 2261-2264.
  22. R. F. Chang, W. T. Chen, and J. S. Wang, "A fast finite-state algorithm for vector quantizer design," *IEEE Transactions on Signal Processing*, Vol. 40, 1992, pp. 221-225.

**Yu-Len Huang (黃育仁)** was born in Chiayi, Taiwan, on May 22, 1970. He received the B.S. degree in Computer Science from Tunghai University, Taiwan, R.O.C., in 1992, and the M.S. and Ph.D. degrees in Computer Science and Information Engineering from National Chung Cheng University, Taiwan, R.O.C., in 1994 and 1999. He is currently an Assistant Professor in the Department of Computer Science and Information Engineering, Tunghai University, Taiwan, R.O.C.. His research interests include digital image/video coding, neural networking, computer networking, and medical imaging. Dr. Huang is a member of IEEE and Phi Tau Phi.

**Ruey-Feng Chang (張瑞峰)** was born in Taichung, Taiwan, on August 25, 1962. He received the B.S. degree in Electrical Engineering from National Cheng Kung University, Tainan, Taiwan, R.O.C., in 1984, and the M.S. degree in Computer and Decision Sciences and the Ph.D. degree in Computer Science from National Tsing Hua University, Hsinchu, Taiwan, R.O.C., in 1988 and 1992, respectively. Since 1992, he has been with the Department of Computer Science and Information Engineering, National Chung Cheng University, Chiayi, Taiwan, R.O.C., and he is now a Professor. His research interests include image/video processing and retrieval, medical computer-aided diagnosis system, and multimedia systems and communication. Dr. Chang is a member of IEEE, ACM, SPIE, and Phi Tau Phi.