# K-means-based color palette design scheme with the use of stable flags

**Article** *in* Journal of Electronic Imaging · July 2007

DOI: 10.1117/1.2762241 · Source: DBLP



**2 authors**, including:

Yu-Chen Hu
Tunghai University, TaiChung City, Taiwan
**258** PUBLICATIONS **4,927** CITATIONS

SEE PROFILE

# K-means-based color palette design scheme with the use of stable flags

**Yu-Chen Hu**
Providence University
Department of Computer Science and Information Engineering
Taichung, Taiwan 433
E-mail: ychu@pu.edu.tw

**Ming-Gong Lee**
Providence University
Department of Computer Science and Information Management
Taichung, Taiwan 433

**Abstract.** *We propose a fast palette design scheme based on the K-means algorithm for color image quantization. To accelerate the K-means algorithm for palette design, the use of stable flags for palette entries is introduced. If the squared Euclidean distances incurred by the same palette entry in two successive rounds are quite similar, the palette entry is classified to be stable. The clustering process will not work on these stable palette entries to cut down the required computational cost. The experimental results reveal that the proposed algorithm consumes a lower computational cost than those comparative schemes while keeping approximately the same image quality.* © *2007 SPIE and IS&T.* [DOI: 10.1117/1.2762241]

## 1 Introduction

Due to the limitations of image display, e.g., data storage and transmission bandwidth, the need for color image compression is essential. Typically, each pixel in the RGB color image consists of three components: red, green, and blue. Each of the color components of one color pixel is represented with 8 bits. In other words, there are 16.8 million possible colors in it. Therefore, for an uncompressed color image of size $512 \times 512$ pixels, the file size is 786 KB.

To reduce the storage cost of the RGB color image, the color image quantization (CIQ) technique[1] is thus introduced. The CIQ technique consists of three procedures: palette design, image encoding, and image decoding. The goal of the palette design procedure is to generate a set of representative color pixels, also called the color palette, that will be used in the image encoding and decoding procedures.

In the CIQ image encoding procedure, each color pixel in the RGB image is processed sequentially. The index of the closest color in the palette is searched and used to encode the input color pixel. The compressed codes are the indices of the closest colors in the palette. Therefore, the compressed image of CIQ is often called the indexed color image or the paletted image. If this image is quantized into the indexed color image using a color palette of $k$ colors, the required storage cost for each index is only $\lceil \log_2 k \rceil$ bits. For example, when a color palette of 256 colors is used in CIQ, the size of compressed file is reduced to about one-third of original image size. Not only is the storage cost reduced, but the transmission time is also reduced when the color image quantization technique is used.

In the image decoding procedure of CIQ, the same palette is stored and used. Each compressed color pixel is recovered by the palette color that corresponds to the received index. By sequentially reconstructing each compressed pixel, the whole compressed color image is then generated. The image decoding procedure is quite simple and requires little computational cost.

Untill now, several palette design algorithms have been proposed. They can be classified into two categories: splitting-based algorithms and clustering-based algorithms. The splitting-based algorithms generally split the color space of the color image into two disjoint groups according to their splitting criteria. Then the splitting procedure is iterated until the desired number of groups is obtained. Finally, the centroid of each group is chosen as the palette color. Examples of the splitting-based algorithms are the median cut method,[2] the mean cut method,[3] the variance-based algorithm (VBA),[4] the radius weighted mean cut method,[5] and the pixel mapping using reduction of color space dimensionality.[6]

The clustering-based algorithms[7–14] group the color spaces into $k$ desired clusters to form a color palette of $k$ colors. In clustering-based algorithm, the first step is to generate an initial palette of $k$ colors. Then, by repeatedly performing the clustering process and the centroid updating process, the desired color palette is thus generated. Typically, the termination condition is controlled either by a fixed number of rounds executed or by checking whether the color palette tends to be stable. Examples of clustering-based algorithms are the K-means algorithm,[7] the fuzzy c-means algorithm,[9,10] the genetic c-mean algorithm,[11] and the color finite-state LBG (CFSLBG) algorithm.[14]

The selection of initial colors in the K-means-based al-

Paper 06021RR received Feb. 7, 2006; revised manuscript received Apr. 12, 2007; accepted for publication Apr. 18, 2007; published online Jul. 30, 2007.
1017-9909/2007/16(3)/033003/11/$25.00 © 2007 SPIE and IS&T.

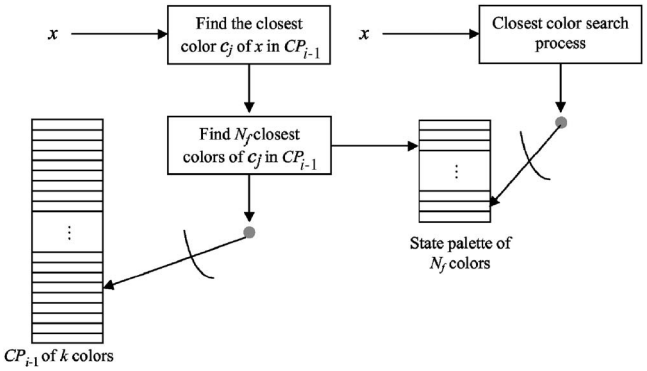



**Fig. 1** Block diagram of generating the state palette in the $i$th iteration ($i \geq 2$) of the CFSLBG algorithm for the training pixel $x$.

gorithms has great influence on the performance of the designed palettes. Some palette initialization methods have been proposed[15,16] for color image quantization. In fact, any splitting algorithms described above can be used as the initial technique to generate the initial palettes. Note that some additional computational cost is consumed when the initial palettes are not randomly selected from the given images. In addition, some methods that aim to accelerate the palette design process and image encoding process have been proposed.[17–19]

In general, the splitting-based algorithms are faster than the clustering-based algorithms. However, they are not guaranteed to obtain an optimal solution because the splitting decision at each level cannot be recovered. By contrast, the clustering-based algorithms can provide a better solution in terms of long execution time.

In this paper, we propose a fast palette design algorithm based on the K-means algorithm. The goal of the proposed scheme is to accelerate the K-means algorithm while keeping approximately the same quality of color palettes. To achieve the goal, an important observation in the pixel grouping process helps us to design the proposed scheme. It is obvious that considerable variations of the palette colors are found in the first few rounds. Then, most training pixels generally are classified into the same entry in the palettes in the successive rounds. Stable checking of palette colors is introduced to avoid the search of closest colors in the palette. That is why it consumes a lower computational cost than the K-means algorithm.

The rest of this paper is organized as follows. We review the K-means algorithm and the CFSLBG algorithm in Section 2. In Section 3, the proposed scheme is introduced. Some experimental results are shown in Section 4 to verify the performance of the proposed scheme. Finally, some conclusions are given in Section 5.

## 2 Related Schemes

In this section, we describe two related palette design algorithms for CIQ. First, the K-means algorithm for palette generation is reviewed. Next, the color finite-state LBG algorithm (CFSLBG) is introduced. These two algorithms belong to the clustering-based algorithms.

### 2.1 *The K-Means Algorithm*

The K-means clustering algorithm[7] was originally proposed for pattern recognition. It is commonly regarded as a suboptimal quantization approach that can also be applied to color vision, image segmentation, and vector quantization (VQ). For image vector quantization, the generalized Lloyd algorithm,[20] also called the LBG algorithm, is equivalent to the K-means algorithm.

The K-means algorithm consists of two main processes: the pixel grouping process and the centroid updating process. By repeating these two processes, the desired color palette is thus obtained. The detailed steps of the K-means algorithms are listed in the following.

**The K-means algorithm:**

Step 1: Generate the initial color palette $CP_0$ of $k$ colors. Set the iteration count $i=1$ and the image distortion $D_0=\infty$.
Step 2: For each training color pixel $x$, find the closest palette color in the current color palette $CP_{i-1}$.
Step 3: Calculate the image distortion $D_i$ incurred in the current iteration.
Step 4: For each entry in $CP_{i-1}$, generate the new palette color by the mean value of the training pixels that belong to this entry.
Step 5: Compute the average distortion ratio $ADR_i$, where $ADR_i=|(D_i-D_{i-1})/D_i|$.
Step 6: If $ADR_i$ is smaller than or equal to the threshold $\varepsilon$, then go to Step 8.
Step 7: Increase $i$ by 1 and go to Step 2.
Step 8: Output the current palette of $k$ colors.

An initial palette of $k$ colors is selected in Step 1. For simplicity, the palette colors can be randomly selected from the training set of image pixels. The randomly selected palette does not guarantee the design of a good color palette. To design a better color palette, one of the splitting-based palette design schemes[2–6] can be employed for palette initialization in Step 1. In Step 2, the closest color in the palette of each training pixel is determined. The image distortion between each training pixel and its closest palette color is accumulated to form the image distortion $D_i$. After Step 2 is executed, these training pixels are classified into $k$ groups. Then the mean value of these pixels in each group is calculated. As a result, $k$ mean values computed form the new palette in Step 4.

The pixel grouping and centroid updating processes are iterated until the incurred distortions of the generated palettes in two consecutive rounds are close. In each round the incurred image distortion is calculated in Step 3. Then the average distortion ratio is computed in Step 5. The definition of the average distortion ratio is given as follows:

$$ADR_i = \left| \frac{D_i - D_{i-1}}{D_i} \right|. \quad (1)$$

Here, $D_i$ and $D_{i-1}$ denote the image distortion incurred in the $i$th and $i-1$th rounds, respectively. The termination condition is checked in Step 6. If $ADR_i$ is smaller than or equal to the threshold $\varepsilon$, the algorithm is then terminated.

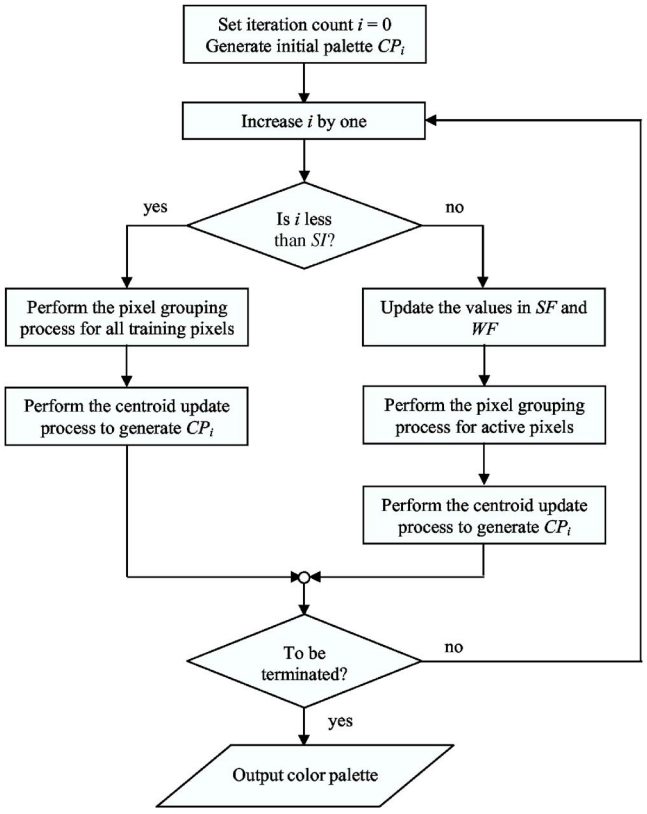



**Fig. 2** Flowchart of the proposed scheme.

Sometimes, another termination condition based on the number of iteration can be used.

### 2.2 *The Color Finite-State LBG Algorithm (CFSLBG)*

In 2003, Huang and Chang proposed a color finite-state LBG (CFSLBG) algorithm[14] that modified the training steps of the ordinary LBG algorithm to speed up the color palette generation process. The LBG algorithm[20] was proposed to design codebooks for image vector quantization. It is a clustering-based algorithm and equivalent to the K-means algorithm. The authors extend the concept of the finite-state vector quantization (FSVQ)[21] that was originally used in vector quantization to color palette design in CIQ.

To design a color palette of $k$ colors using CFSLBG, an initial palette $CP_0$ of $k$ colors is first selected. Then each training pixel is classified into the group corresponding to its closest palette color in $CP_0$. Then, the mean value of the color pixels in each group is calculated to generate the new palette color in $P_1$. The processing steps for generating $CP_0$ and $CP_1$ in CSFLBG are the same as those of the traditional LBG algorithm.

To design the color palette $CP_i$ in CFSLBG when the iteration count $i$ is greater than or equal to 2, the concept of a finite state is used in the clustering process of CFSLBG. To cut down the computational cost needed in the traditional LBG algorithm, a subset of the color palette instead of all the palette colors is searched to find the closest color in the palette of each training pixel in CFSLBG. The subset of the palette is also called the state palette. Let $N_f$ denote the size of the state palette. In CFSLBG, the size of state palette $N_f$ must be defined before executing the palette design process. The block diagram of generating the state palette of $N_f$ color for the training pixel $x$ is depicted in Fig. 1.

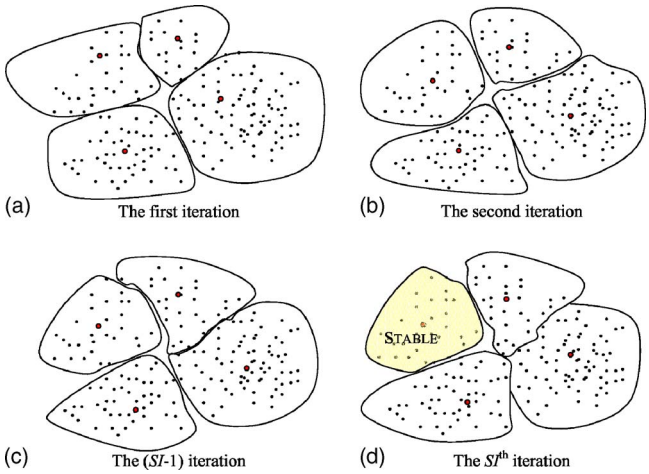



**Fig. 3** Example of the proposed scheme to design a palette of four colors.

Let $CP_i$ denote the color palette that is generated in the $i$th iteration of the CFSLBG algorithm for $i \geq 1$. In CFSLBG, the closest color $c_j$ in the color palette $CP_{i-1}$ of each training pixel $x$ is used to determine the state palette $SP_x$ of $N_f$ colors for $x$ in the $i$th iteration. The state palette $SP_x$ is the subset of the whole palette $P_i$ containing $N_f$ colors that are the closest colors to $c_j$ in $CP_i$. The reason why these $N_f$ closest colors instead of all the colors in the palette are used is to reduce the consumed execution time for the pixel grouping process. The detailed processing steps of CFSLBG are listed in the following.

**The CFSLBG algorithm:**

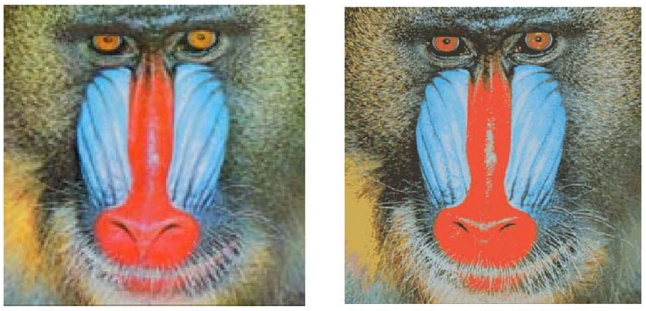
Step 1: Select an initial color palette $CP_0$. Set the iteration count $i=1$ and the image distortion $D_0=\infty$.
Step 2: Find the closest palette color in $CP_0$ for each training pixel $x$. Compute the image distortion $D_1$.
Step 3: For each palette color in $CP_0$, compute the mean value of the training color pixels that are closest to it, and then add the new palette color into $CP_1$. Set $i=2$.
Step 4: For each training pixel $x$, the closest palette color $c_j$ in $CP_{i-1}$ is taken as the state $s$ for $x$. For each state $s$, find $N_f$ closest colors in $CP_{i-1}$ to generate the state palette. Find the closest palette color for $x$ by searching the state palette.
Step 5: For each entry in $CP_{i-1}$, generate the new palette color in $CP_i$ by the mean value of the training pixels that belong to this entry.
Step 6: Compute the average distortion ratio $ADR_i$.
Step 7: If $ADR_i$ is smaller than or equal to the threshold $\varepsilon$, then go to Step 9.
Step 8: Increase $i$ by 1 and go to Step 4.
Step 9: Output the current palette of $k$ colors.

(a) Airplane (b) 16 colors with 30.657 dB

(c) 32 colors with 32.670 dB (d) 64 colors with 36.343 dB

(e) 128 colors with 38.147 dB (f) 256 colors with 39.960 dB

**Fig. 4** Source and resultant images of "Airplane" of the proposed scheme using different palette sizes (color online only).



(a) Source image of "Baboon" (b) 16 colors with 24.344dB

(c) 32 colors with 26.997 dB (d) 64 colors with 28.877 dB

(e) 128 colors with 30.692 dB (f) 256 colors with 32.576 dB

**Fig. 5** Source and resultant images of "Baboon" of the proposed scheme using different palette sizes (color online only).

## 3 The Proposed Scheme

The goal of the proposed scheme is to cut down the computational cost of the K-means algorithm for palette design. Since the K-means algorithm iterates the pixel group and centroid updating processes until the performance of the color palettes of two successive rounds are approximately the same, a great deal of execution time is needed.

An important observation in the pixel grouping process of the K-means algorithm helps us to design the proposed scheme. In the first few rounds considerable variations of the palette colors are found, because the palettes designed are unstable. After the first few rounds, we find that most training pixels are classified into the same entry in the palettes during two successive rounds. In other words, if one training pixel $x$ is classified into the group corresponding to the $j$th color in $i-1$ iterations, it is quite possible that $x$ is also classified into the $j$th group in $i$ iterations. That is because the colors corresponding to the same entry in the palettes of two successive rounds should be quite similar.

In the proposed scheme, the stable flag $SF$ and the working flag $WF$ are used. $SF$ is an array of $k$ entries that is used to record the status of the palette colors during the palette design process. The values in $SF$ can be either stable or unstable. $WF$ is used to record whether each training pixel takes part in the palette design process. The size of $WF$ equals the number of training pixels in the given image. The values in $WF$ can be either active or inactive. Here, an active entry in $WF$ indicates that the corresponding training pixel is used in the design of the color palette. By contrast, one training pixel is not used to design the color palette if the corresponding entry in $WF$ is set to be inactive. Initially, all entries in $SF$ and $WF$ are set to unstable and active, respectively.

Three control threshold values are used in the proposed scheme. A predefined starting iteration $SI$ is used to start the use of the stable flags. Note that $SI$ should be greater than or equal to 2. In addition, a predefined threshold $\theta$ is used to control the similarity between two given colors. Further, the threshold $\varepsilon$ is used to control the termination condition. The termination condition of the proposed scheme is the same to that used in the K-means algorithm.

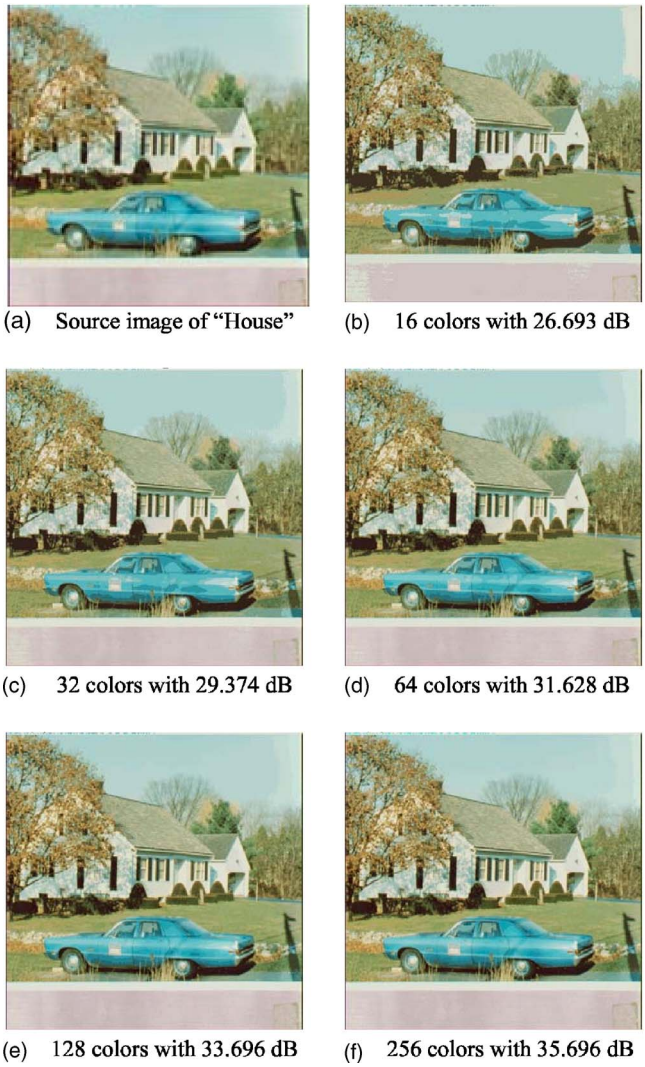



**Fig. 6** Source and resultant images of "House" of the proposed scheme using different palette sizes (color online only).


**Fig. 7** Source and resultant images of "Lena" of the proposed scheme using different palette sizes (color online only).

The flowchart of the proposed scheme is given in Fig. 2. The detailed processing steps are given in the following.

**The proposed algorithm:**

Step 1: Select an initial color palette $CP_0$ of $k$ colors. Set the image distortion $D_0=\infty$ and the iteration count $i=1$. Set all entries in $SF$ and $WF$ to be unstable and active, respectively.

Step 2: If $i$ is less than the starting iteration $SI$, the following substeps are iterated.

Step 2.1: Find the closest palette color in $CP_i$ for each training pixel $x$. Classify $x$ into the specific group that corresponds to its closest palette color.

Step 2.2: Compute the mean value of the training color pixels in each group. These $k$ mean values form the color palette $CP_i$. Go to Step 8.

Step 3: For each unstable entry $j$ in $SF$, the following substeps are executed:

Step 3.1: Compute the squared Euclidean distance $d(pc_j^{i-2}, pc_j^{i-1})$ between the $j$th palette colors in the two previous rounds.

Step 3.2: If $d(pc_j^{i-2}, pc_j^{i-1})$ is less than or equal to $\theta$, the $j$th entry in $SF$ is set to be stable.

Step 4: For each entry in $CP_i$ that has a corresponding stable entry in $SF$, generate the palette color in $CP_i$ by using the color in $CP_{i-1}$.

Step 5: For each training pixel $x$, get the index $idx$ of the closest color in $CP_{i-1}$. If the $idx$th entry in $SF$ is stable, the corresponding entry of $x$ in $WF$ is set to inactive.

Step 6: For each training pixel $x$ whose corresponding entry in $WF$ is active, find the closest color of $x$ from these unstable colors in $CP_{i-1}$. Classify $x$ into the specific group that corresponds to its closest palette color.

Step 7: For each entry in $CP_{i-1}$ that corresponds to an unstable entry in $SF$, generate the new

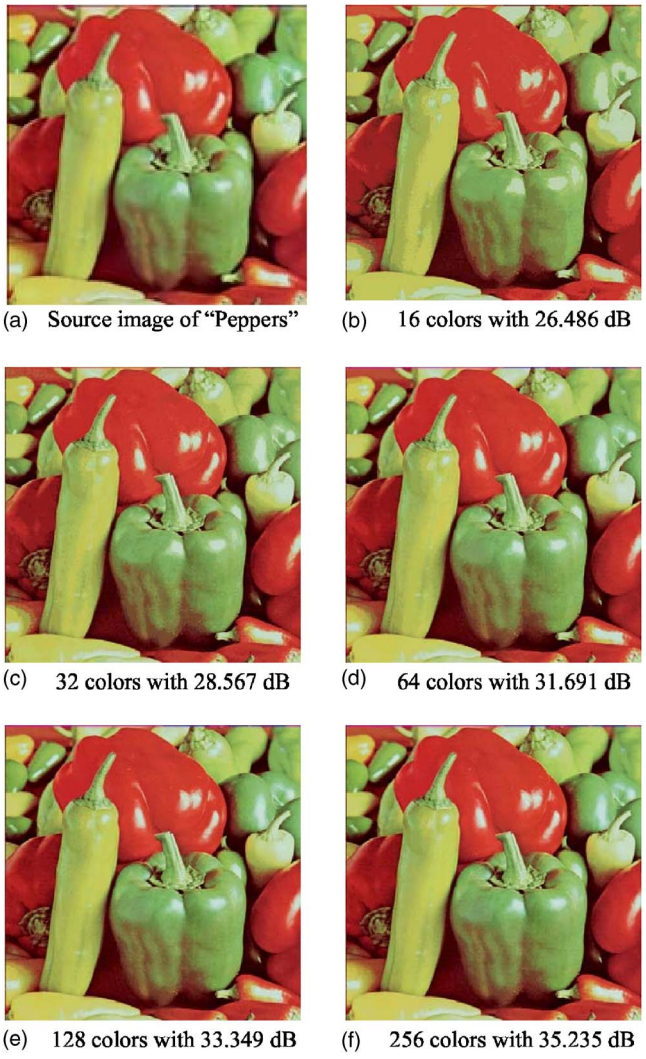



**Fig. 8** Source and resultant images of "Peppers" of the proposed scheme using different palette sizes (color online only).

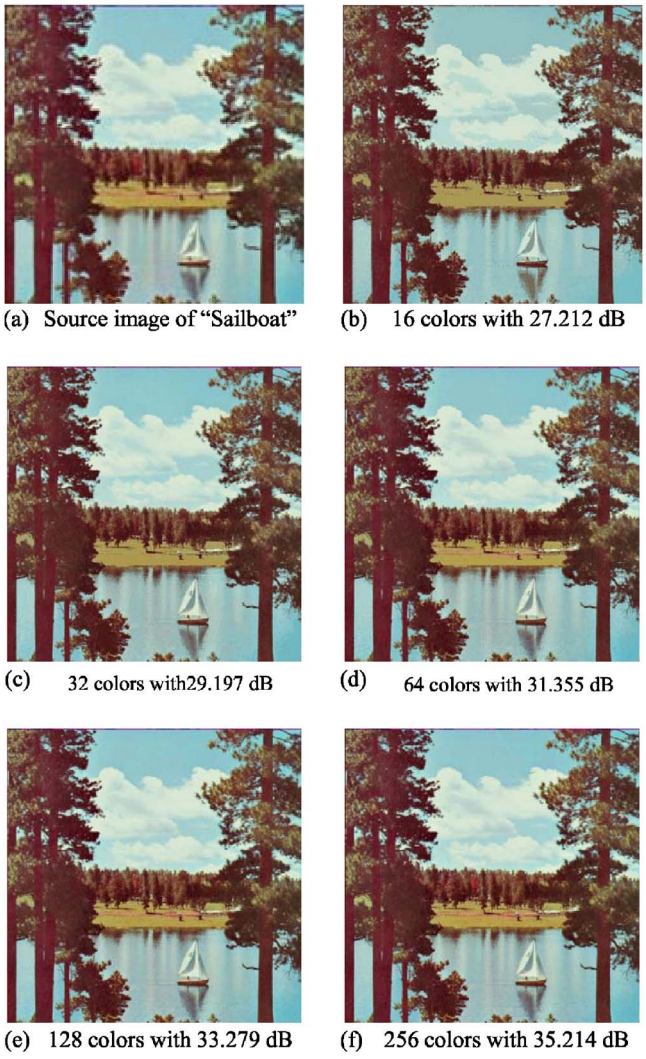



**Fig. 9** Source and resultant images of "Sailboat" of the proposed scheme using different palette sizes (color online only).

palette color in $CP_i$ by the mean value of the pixels.

Step 8: Compute the image distortion $D_i$ and then calculate the average distortion $ADR_i$. If $ADR_i$ is smaller than or equal to the threshold $\varepsilon$, then go to Step 10.

Step 9: Increase $i$ by 1 and go to Step 2.

Step 10: Output the current palette of $k$ colors.

The processing substeps of designing the color palettes in the first $(SI-1)$ rounds are the same as those of the traditional K-means algorithm. The value of $SI$ should be greater than or equal to 2 in the proposed algorithm. To design the color palettes in the successive rounds, an array $SF$ of $k$ entries is used. Initially, each palette entry in $SF$ is set to unstable in Step 1. The entries of $SF$ are updated in Step 3 if the iteration count is greater than or equal to $SI$. Here, a simple rule is used to determine whether or not one unstable entry in $SF$ is to be updated. If the squared Euclidean distance between the corresponding palette colors $pc_j^{i-2}$ and $pc_j^{i-1}$ in the two previous rounds, respectively, are quite close, the $j$th entry in the color palette is set to be stable.

**Table 1** Results of the image quality of the LBG and CFSLBG with different palette sizes.

| Palette Sizes | K-Means Algorithm | CFSLBG | | | |
|---|---|---|---|---|---|
| | | $N_f$=4 | $N_f$=8 | $N_f$=12 | $N_f$=16 |
| 16 | 27.883 | 27.871 | 27.901 | 27.884 | 27.884 |
| 32 | 30.392 | 30.280 | 30.344 | 30.382 | 30.378 |
| 64 | 32.485 | 32.316 | 32.541 | 32.572 | 32.559 |
| 128 | 34.291 | 34.336 | 34.531 | 34.254 | 34.382 |
| 256 | 36.830 | 36.596 | 36.712 | 36.378 | 36.867 |

**Table 2** Average results of the image quality of the proposed scheme with different palette sizes and different *SI* values.

| Palette Sizes | *SI* | The Proposed Scheme | | | | |
|---|---|---|---|---|---|---|
| | | $\theta$=1 | $\theta$=2 | $\theta$=3 | $\theta$=4 | $\theta$=5 |
| 16 | 2 | **27.786** | 27.687 | 27.672 | 27.663 | 27.626 |
| | 4 | **27.814** | 27.790 | 27.781 | 27.777 | 27.772 |
| | 6 | **27.845** | 27.833 | 27.826 | 27.825 | 27.821 |
| | 8 | **27.857** | 27.851 | 27.849 | 27.849 | 27.847 |
| | 10 | **27.873** | 27.866 | 27.865 | 27.865 | 27.864 |
| 32 | 2 | **30.200** | 30.111 | 30.094 | 30.087 | 29.857 |
| | 4 | **30.236** | 30.201 | 30.199 | 30.192 | 29.973 |
| | 6 | **30.294** | 30.268 | 30.266 | 30.260 | 30.249 |
| | 8 | **30.316** | 30.304 | 30.301 | 30.300 | 30.299 |
| | 10 | **30.334** | 30.329 | 30.327 | 30.326 | 30.325 |
| 64 | 2 | **32.370** | 32.292 | 32.266 | 32.255 | 32.233 |
| | 4 | **32.441** | 32.416 | 32.408 | 32.399 | 32.395 |
| | 6 | **32.487** | 32.475 | 32.472 | 32.471 | 32.469 |
| | 8 | **32.525** | 32.512 | 32.511 | 32.511 | 32.509 |
| | 10 | **32.542** | 32.533 | 32.532 | 32.532 | 32.531 |
| 128 | 2 | **34.353** | 34.293 | 34.274 | 34.268 | 34.250 |
| | 4 | **34.445** | 34.417 | 34.410 | 34.408 | 34.403 |
| | 6 | **34.501** | 34.487 | 34.484 | 34.482 | 34.481 |
| | 8 | **34.533** | 34.531 | 34.530 | 34.529 | 34.528 |
| | 10 | 34.552 | 34.553 | **34.554** | 34.553 | 34.552 |
| 256 | 2 | **36.246** | 36.191 | 36.172 | 36.163 | 36.147 |
| | 4 | **36.343** | 36.324 | 36.322 | 36.319 | 36.314 |
| | 6 | **36.398** | 36.390 | 36.392 | 36.391 | 36.390 |
| | 8 | 36.424 | 36.428 | **36.429** | **36.429** | 36.428 |
| | 10 | 36.446 | 36.454 | **36.456** | 36.454 | 36.453 |

The squared Euclidean distance between $pc_j^{i-2}$ and $pc_j^{i-1}$ can be calculated according to the following equation

$$d(pc_j^{i-2}, pc_j^{i-1}) = \sum_{t=1}^{3} (pc_j^{i-2}(t) - pc_j^{i-1}(t))^2. \quad (2)$$

If the calculated distance is less than or equal to a predefined threshold $\theta$, the $j$th entry in *SF* is set to be stable. Otherwise, it is still in an unstable state. Here, the threshold $\theta$ is a predefined value that is used to control the degree of similarity between two given color pixels. In Eq. (2), $pc_j^i$ denotes the $j$th palette color in the $i$th iteration. For example, $pc_{11}^1$ is the 11th palette color in the palette $CP_1$.

**Table 3** Results of the execution time (unit: seconds) of the K-means algorithm and CFSLBG with different palette sizes.

| Palette Sizes | K-Means Algorithm | CFSLBG | | | |
|---|---|---|---|---|---|
| | | $N_f$=4 | $N_f$=8 | $N_f$=12 | $N_f$=16 |
| 16 | 6.603 | 4.833 | 6.828 | 7.747 | 9.644 |
| 32 | 20.433 | 8.411 | 9.947 | 14.022 | 16.883 |
| 64 | 51.136 | 7.281 | 11.542 | 15.525 | 21.092 |
| 128 | 83.103 | 8.750 | 12.231 | 38.731 | 45.164 |
| 256 | 94.642 | 10.678 | 19.947 | 51.181 | 21.983 |

In Step 4, the palette color $pc_j^i$ in $CP_i$ is duplicated to generate the palette color $pc_j^i$ in $CP_i$ for each stable entry $j$ in *SF*. In other words, these training pixels that are classified into one stable palette color in the previous rounds are not included for the palette design in the current iteration. If an adequate control threshold $\theta$ is set, we assume that the further designed $pc_j^i$ should be quite close to the palette color $pc_j^{i-1}$ that is generated in the pervious iteration. In other words, $pc_j^{i-1}$ is taken as the approximate value for $pc_j^i$.

If one entry in *SF* is set to be stable, the palette color corresponding to it is assumed to be quite good to represent these training pixels that are closest to this palette color. Therefore, these training pixels will not be used to design the color palette in the successive rounds. The entries in *WF* of these pixels are set to be inactive in Step 5.

In Steps 6 and 7, these training pixels that have active entries in *WF* are included to design the palette colors that are still unstable. Only the palette colors having unstable entries in *SF* are taken as candidates in the closest-color search process for each active training pixel. Compared to the pixel grouping process of the K-means algorithm, the consumed computational cost in Step 6 of the proposed scheme is much lower than that of the K-means algorithm.

In Step 8, the image distortion incurred in each iteration is calculated, and in turn the average distortion ratio is computed. The average distortion ratio can be calculated by using Eq. (1). The termination condition of the proposed algorithm is checked in Step 9. If the average distortion ratio is smaller than or equal to the threshold $\varepsilon$, the proposed algorithm is then terminated.

Remember that all the inactive training pixels do not take part in the palette design process when the iteration number is greater than or equal to *SI*. That is because we assume that the corresponding palette colors provide good representative of them. Only these active training pixels are used to design some palette colors in Steps 6 and 7. Obviously, the number of inactive training pixels decreases as the iteration count increases. That is because more stable entries are set in *SF* as the proposed scheme keeps going. That is why its computational cost can be significantly reduced.

**Table 4** Average results of the execution time (unit: seconds) of the proposed scheme with different palette sizes and different *SI* values.

| Palette Sizes | *SI* | The Proposed Scheme | | | | |
|---|---|---|---|---|---|---|
| | | $\theta=1$ | $\theta=2$ | $\theta=3$ | $\theta=4$ | $\theta=5$ |
| 16 | 2 | 1.222 | 0.928 | 0.869 | 0.836 | 0.769 |
| | 4 | 1.253 | 0.992 | 0.933 | 0.911 | 0.875 |
| | 6 | 1.381 | 1.172 | 1.100 | 1.092 | 1.069 |
| | 8 | 1.503 | 1.397 | 1.381 | 1.381 | 1.375 |
| | 10 | 1.817 | 1.675 | 1.661 | 1.656 | 1.636 |
| 32 | 2 | 1.389 | 1.067 | 1.033 | 1.019 | 0.922 |
| | 4 | 1.411 | 1.206 | 1.206 | 1.183 | 1.108 |
| | 6 | 1.711 | 1.444 | 1.417 | 1.406 | 1.394 |
| | 8 | 1.953 | 1.756 | 1.725 | 1.717 | 1.703 |
| | 10 | 2.200 | 2.075 | 2.064 | 2.061 | 2.050 |
| 64 | 2 | 1.586 | 1.108 | 1.008 | 0.958 | 0.914 |
| | 4 | 1.806 | 1.572 | 1.519 | 1.486 | 1.442 |
| | 6 | 2.206 | 2.025 | 1.944 | 1.931 | 1.936 |
| | 8 | 2.800 | 2.519 | 2.450 | 2.458 | 2.464 |
| | 10 | 3.322 | 2.994 | 2.975 | 2.978 | 2.967 |
| 128 | 2 | 1.697 | 1.400 | 1.347 | 1.297 | 1.247 |
| | 4 | 2.178 | 1.903 | 1.833 | 1.808 | 1.772 |
| | 6 | 2.806 | 2.514 | 2.494 | 2.472 | 2.453 |
| | 8 | 3.408 | 3.194 | 3.125 | 3.136 | 3.114 |
| | 10 | 3.989 | 3.869 | 3.833 | 3.811 | 3.825 |
| 256 | 2 | 1.975 | 1.564 | 1.494 | 1.478 | 1.400 |
| | 4 | 2.717 | 2.356 | 2.331 | 2.297 | 2.269 |
| | 6 | 3.533 | 3.272 | 3.222 | 3.217 | 3.192 |
| | 8 | 4.419 | 4.211 | 4.158 | 4.150 | 4.139 |
| | 10 | 5.356 | 5.169 | 5.122 | 5.125 | 5.119 |

**Table 5** Results of the iteration number needed in the K-means algorithm and CFSLBG with different palette sizes.

| Palette Sizes | K-Means Algorithm | CFSLBG | | | |
|---|---|---|---|---|---|
| | | $N_f=4$ | $N_f=8$ | $N_f=12$ | $N_f=16$ |
| 16 | 14.167 | 21.167 | 18.167 | 16.500 | 14.000 |
| 32 | 24.167 | 32.500 | 29.833 | 22.333 | 23.500 |
| 64 | 33.167 | 23.333 | 21.833 | 23.333 | 23.667 |
| 128 | 62.333 | 106.833 | 21.500 | 21.000 | 61.167 |
| 256 | 16.333 | 75.000 | 17.500 | 30.667 | 75.667 |

An example of the proposed algorithm is depicted in Fig. 3. Suppose a color palette of four colors is to be designed. The initial palette is first selected, and the pixel grouping result of the first iteration is given in Fig. 3(a). Similarly, the pixel grouping result of the second iteration is listed in Fig. 3(b). From Figs. 3(a) and 2(b), we find that quite a few color pixels are partitioned into different groups in these two rounds. Continuing this example, the pixel grouping result of the $(SI-1)$th iteration of the proposed algorithm is depicted in Fig. 3(c). To design the color palette in the $SI$th iteration, Step 3 is executed. Suppose one of the four palette colors is determined to be stable, as shown in Fig. 3(d); then, these training pixels that were grouped to it in the $(SI-1)$th iteration are not included in the design of the palette colors in the $SI$th iteration. That is why the consumed computational cost of the proposed algorithm can be reduced compared to that of the K-means algorithm.

## 4 Experimental Results

To verify the performance of the proposed schemes, a variety of simulations had been performed on a personal computer (PC) Pentium 4 2.4 GHz with 256 MB RAM using the Linux RedHat 9.0 operating system. Six testing color images—"Airplane," "Baboon," "House," "Lena," "Pepper," and "Sailboat," as shown in part (a) of Figs. 4–9—were used in the simulations. Each of the testing image is of size $512\times512$ pixels.

In the simulations, palettes of size 16, 32, 64, 128, and 256 colors were designed. In the simulations, the colors in the initial palette were selected from the diagonal line of the image with the limitation that color pixels with the same values only appear once. In addition, $\varepsilon$ is set to 0.0001 for controlling the termination of the K-means algorithm.

To measure the image quality of the compressed image, the peak signal-to-noise ratio (PSNR) between the original image and the compressed image is calculated as

$$\mathrm{PSNR}=10\times\log_{10}\frac{255^2}{\mathrm{MSE}}\mathrm{dB}. \tag{3}$$

Note that MSE denotes the mean-squared error between the original and compressed images. Note that PSNR is generally considered an indication of image quality rather than a definitive calculation. However, it is a commonly used measurement for evaluating the image quality. Typically, a large PSNR value indicates that the difference between two given images is quite small. From the literature, when the PNSR value between two given images is greater than or equal to 30 dB, it is assumed that these two images are quite similar and it is hard to distinguish them via human visual systems (HVS).

Comparative results of the image quality of the K-means algorithm and CFSLBG are listed in Table 1. In the CFS-

**Table 6** Average results of the iteration number needed in the proposed scheme with different palette sizes and different *SI* values.

| Palette Sizes | *SI* | The Proposed Scheme | | | | |
|---|---|---|---|---|---|---|
| | | $\theta$=1 | $\theta$=2 | $\theta$=3 | $\theta$=4 | $\theta$=5 |
| 16 | 2 | 9.833 | 8.167 | 7.833 | 7.500 | 7.167 |
| | 4 | 10.333 | 8.000 | 7.667 | 7.500 | 7.167 |
| | 6 | 10.833 | 9.000 | 8.333 | 8.333 | 8.167 |
| | 8 | 11.333 | 10.333 | 10.333 | 10.333 | 10.167 |
| | 10 | 13.667 | 12.167 | 12.167 | 12.167 | 12.000 |
| 32 | 2 | 10.333 | 8.667 | 8.667 | 8.667 | 7.833 |
| | 4 | 10.000 | 9.000 | 9.000 | 9.000 | 8.000 |
| | 6 | 11.667 | 9.667 | 9.333 | 9.333 | 9.167 |
| | 8 | 12.667 | 11.000 | 10.667 | 10.667 | 10.500 |
| | 10 | 13.500 | 12.333 | 12.333 | 12.333 | 12.167 |
| 64 | 2 | 9.833 | 6.500 | 6.000 | 5.500 | 5.333 |
| | 4 | 9.833 | 8.833 | 8.500 | 8.167 | 7.833 |
| | 6 | 10.667 | 9.833 | 9.167 | 9.000 | 9.167 |
| | 8 | 13.667 | 11.167 | 10.667 | 10.667 | 10.667 |
| | 10 | 15.500 | 12.333 | 12.167 | 12.167 | 12.167 |
| 128 | 2 | 9.333 | 8.667 | 8.667 | 8.167 | 7.833 |
| | 4 | 10.000 | 8.833 | 8.500 | 8.000 | 7.667 |
| | 6 | 11.333 | 9.833 | 9.667 | 9.500 | 9.333 |
| | 8 | 12.333 | 11.167 | 10.667 | 10.667 | 10.667 |
| | 10 | 13.000 | 12.833 | 12.500 | 12.500 | 12.500 |
| 256 | 2 | 8.333 | 7.000 | 7.167 | 7.167 | 6.500 |
| | 4 | 9.167 | 7.667 | 7.833 | 7.500 | 7.167 |
| | 6 | 10.000 | 9.000 | 8.833 | 8.833 | 8.667 |
| | 8 | 11.167 | 10.833 | 10.500 | 10.500 | 10.000 |
| | 10 | 13.167 | 12.667 | 12.500 | 12.500 | 12.333 |

LBG algorithm, the sizes of state palette $N_f$ are set to 4, 8, 12, and 16. From the results, CFSLBG provides slightly better image quality than that of the K-means algorithm in most cases except for the design of color palette of 32 colors. In general, the image quality increases as the increment of the palette size in CFSLBG.

Table 2 shows the average results of the image quality of the proposed scheme. Here, different *SI* values are used. In addition, the controlling threshold $\theta$ values ranging from 1 to 5 are used in the simulations. According to the results, we find that a better image quality is obtained when a smaller $\theta$ value is used and the value of *SI* is held constant, because a large portion of the training pixels is involved in the pixel grouping process when a small $\theta$ value is used.

In addition, the image quality increases as the increment of the value of *SI* when the $\theta$ value is held constant. In fact, the performance of the proposed scheme is equivalent to the K-means algorithm when the value of *SI* is larger than the iteration number needed in the K-means algorithm. According to the results, we suggest that the $\theta$ value be set to 1.

Compared to the results of the K-means algorithm in Table 1, the proposed scheme has approximately the same performance with the K-means algorithm in the design of 16- and 32-color palettes. It designs better palettes of 64 colors than the K-means algorithm when the values of *SI* are greater than or equal to 6. In addition, it designs better palettes of 128 colors than the K-means algorithm. A slightly worse image quality of the proposed scheme is obtained compared to that of the K-means algorithm when the palette size is 256.

Table 3 lists the results of the computational cost needed in the K-means algorithm and CFSLBG. A higher computational cost results when a large-sized state palette is used in CFSLBG. In most cases, CFSLBG consumes a lower computational cost than that of the K-means algorithm except for the design of color palettes of 16 colors.

Experimental results of the execution time of the proposed scheme are given in Table 4. Its required execution time increases when a large *SI* value is used. The execution time decreases as the $\theta$ value increases. In Table 3, average execution times of 51.136 and 94.642 s are needed in the K-means algorithm to design the color palettes of 64 and 256 colors, respectively. It requires 7.80% of the execution time of the K-means algorithm in design color palettes of 64 colors when $\theta$ and *SI* equal 1 and 10, respectively. Moreover, only 5.66% of the execution time is consumed to design a 256-color palette when $\theta$ and *SI* equal 1 and 10, respectively. In other words, the proposed scheme significantly reduces the computational cost.

Table 5 shows the results of the iteration number needed in the K-means algorithm and CFSLBG. We find that the iteration number decreases when a large-sized state palette is used in CFSLBG. It is interesting that the number of iterations needed to design the color palette of 128 colors for the testing image "Lena" is much higher than those for other testing images. A similar problem can also be observed in the design of 128- and 256-color palettes for the testing image "Airplane." It may be because some similar palette colors are generated during the design of the color palette. Some training pixels are classified into these similar palette colors in two or more successive rounds so that the required iteration number increases.

The average results of the iteration number of the proposed scheme are given in Table 6. We find that the iteration number needed decreases as the increment of $\theta$ value. In addition, the required iteration number increases as the increment of the value of *SI*. But the required iteration numbers of the proposed scheme are much smaller than those of the K-means algorithm and CFSLBG, because the use of stable flags significantly reduces the number of training pixels needed to be clustered in the proposed scheme.

**Table 7** Results of percentage of stable colors (PSC) and percentage of inactive training pixels (PITP) in different iteration of the proposed scheme when $\theta$ and $SI$ are set to 1 and 2, respectively.

| | | 2 | | 4 | | 6 | | 8 | |
|---|---|---|---|---|---|---|---|---|---|
| Palette Sizes | Images | PSC | PITP | PSC | PITP | PSC | PITP | PSC | PITP |
| 16 | Airplane | 6.25% | 4.02% | 37.50% | 57.68% | 50.00% | 66.42% | 56.25% | 72.09% |
| | Baboon | 0.00% | 0.00% | 18.75% | 25.33% | 43.75% | 57.62% | 43.75% | 56.91% |
| | House | 0.00% | 0.00% | 18.75% | 37.91% | 25.00% | 41.26% | 56.25% | 60.55% |
| | Lena | 0.00% | 0.00% | 18.75% | 32.39% | 43.75% | 56.72% | 50.00% | 59.54% |
| | Pepper | 0.00% | 0.00% | 25.00% | 35.26% | 37.50% | 43.29% | 75.00% | 79.89% |
| | Sailboat | 0.00% | 0.00% | 12.50% | 12.17% | 68.75% | 69.35% | 87.50% | 88.90% |
| 32 | Airplane | 9.38% | 8.58% | 43.75% | 63.14% | 65.63% | 78.41% | 78.13% | 88.45% |
| | Baboon | 0.00% | 0.00% | 12.50% | 12.04% | 31.25% | 33.37% | 53.13% | 56.32% |
| | House | 0.00% | 0.00% | 21.88% | 33.84% | 46.88% | 57.65% | 62.50% | 71.63% |
| | Lena | 0.00% | 0.00% | 28.13% | 30.61% | 59.38% | 71.27% | 84.38% | 89.39% |
| | Pepper | 3.13% | 0.50% | 25.00% | 31.25% | 65.63% | 69.46% | 93.75% | 95.70% |
| | Sailboat | 0.00% | 0.00% | 18.75% | 25.05% | 34.38% | 41.61% | 59.38% | 69.45% |
| 64 | Airplane | 7.81% | 13.55% | 37.50% | 57.56% | 65.63% | 81.36% | 81.25% | 89.95% |
| | Baboon | 0.00% | 0.00% | 18.75% | 18.27% | 48.44% | 55.51% | 76.56% | 83.29% |
| | House | 3.13% | 0.21% | 21.88% | 24.10% | 50.00% | 57.12% | 68.75% | 75.77% |
| | Lena | 1.56% | 1.17% | 46.88% | 57.48% | 81.25% | 87.40% | 96.88% | 98.23% |
| | Pepper | 3.13% | 1.50% | 26.56% | 30.84% | 67.19% | 70.68% | 84.38% | 86.43% |
| | Sailboat | 1.56% | 1.24% | 26.56% | 29.04% | 56.25% | 62.09% | 73.44% | 80.68% |
| 128 | Airplane | 16.41% | 24.67% | 53.91% | 75.56% | 66.41% | 83.21% | 82.81% | 90.88% |
| | Baboon | 3.91% | 3.57% | 28.13% | 30.79% | 54.69% | 60.46% | 78.91% | 83.69% |
| | House | 4.69% | 1.88% | 29.69% | 43.26% | 60.16% | 67.74% | 77.34% | 81.79% |
| | Lena | 7.03% | 4.75% | 51.56% | 55.40% | 83.59% | 88.37% | 96.88% | 97.58% |
| | Pepper | 4.69% | 4.98% | 40.63% | 44.49% | 65.63% | 69.99% | 84.38% | 89.09% |
| | Sailboat | 3.91% | 4.01% | 30.47% | 41.74% | 61.72% | 71.36% | 82.03% | 86.18% |
| 256 | Airplane | 21.48% | 35.65% | 58.98% | 78.72% | 80.08% | 89.77% | 86.33% | 92.69% |
| | Baboon | 3.91% | 3.45% | 30.47% | 32.76% | 63.67% | 68.39% | 83.98% | 86.68% |
| | House | 6.64% | 3.73% | 40.23% | 52.90% | 73.05% | 80.39% | 88.67% | 91.25% |
| | Lena | 14.06% | 10.65% | 61.33% | 64.71% | 91.02% | 93.93% | 98.44% | 99.15% |
| | Pepper | 6.64% | 7.04% | 41.02% | 47.24% | 74.22% | 78.60% | 94.92% | 96.90% |
| | Sailboat | 3.91% | 4.08% | 35.94% | 45.84% | 73.44% | 82.25% | 91.02% | 94.75% |

To understand the image dependence among the test images, the results of the percentage of stable colors and the percentage of inactive training pixels in different rounds of the proposed algorithm are given in Table 7. Here, the values of $\theta$ and $SI$ are set to 1 and 2, respectively. The percentage of stable colors and the percentage of inactive training pixels increase as the increment of the iteration number when the palette size is held constant. Also, the percentage of stable colors and the percentage of inactive training pixels increase as the increment of palette size. Among these six images, the percentage of stable colors of the low-detailed image "Airplane" is much higher than that of the high-detailed image "Baboon." It is obvious that the total number of rounds executed to design a color palette using "Airplane" is less than that used for "Baboon."

To understand the visual quality of the proposed scheme, source images and compressed images of six testing images are listed in Figs. 4–9. Differently sized palettes are used to show the visual differences in the compressed images. Here, $\theta$ and $SI$ are set to 1 and 10, respectively. According to the results, we find that the compressed images are quite good when the palettes sizes are greater than or equal to 128.

## 5 Conclusions

We proposed a novel palette design algorithm for CIQ. This simple and efficient algorithm works on the RGB color model to design color palettes of different sizes at a low computational cost. As we know the RGB color model is a simple way to represent colors, but a high degree of correlation can be found in its color spectrum. The use of stable flags for palette entries is introduced to decrease the computational cost needed in the traditional K-means algorithm. From the results, we find that it provides approximately the same image quality as the K-means algorithm. Compared to it, about 5.66% of the execution time is consumed to design a 256-color palette when $\theta$ and $SI$ equal 1 and 10, respectively. In other words, a great deal of computational cost can be saved in the proposed scheme. To further improve the proposed algorithm, the applications of the perceptual spaces and metrics[22–24] such as CIELAB and S-CIELAB will be exploited. S-CIELAB is an extension of the CIELAB color difference formula. We will try to find a good way to design color palettes based on the perceptual spaces in the future.

### Acknowledgment

This research was supported by the National Science Council, Taipei, R.O.C., under contract NSC 94-2213-E-126-009.

### References

1. J. P. Allebach, "Digital color imaging bring color to the desktop," *IEEE SP-Society Distinguished Lecture's Series: Third Signal Processing Workshop*, Goreme, Turkey (1995).
2. P. Heckbert, "Color image quantization for frame buffer display," *Comput. Graph.* **16**, 297–307 (1982).
3. X. Wu and I. H. Witten, "A fast K-means type clustering algorithm," Technical report, Department of Computer Science, University of Calgary, Calgary, Canada.
4. S. J. Wan, P. Prusinkiewicz, and S. K. M. Wong, "Variance-based color image quantization for frame buffer display," *Color Res. Appl.* **15**, 52–58 (1990).
5. C. Y. Yang and J. C. Lin, "RWM-cut for color image quantization," *Comput. Graph.* **20**(4), 577–588 (1996).
6. S. C. Cheng and C. K. Yang, "A fast novel technique for color quantization using reduction of color space dimensionality," *Pattern Recogn. Lett.* **22**, 845–856 (2001).
7. J. T. Tou and R. C. Gonzalez, *Pattern Recognition Principles*, Addison-Wesley, Reading, MA (1974).
8. P. Scheunders, "A comparison of clustering algorithms applied to color image quantization," *Pattern Recogn. Lett.* **18**, 1379–1384 (1997).
9. Y. W. Lim and S. U. Lee, "On the color image segmentation algorithm based on the thresholding and the fuzzy C-means technique," *Pattern Recogn.* **23**, 935–952 (1990).
10. D. Ozdemir and L. Akarun, "A fuzzy algorithm for color quantization of images," *Pattern Recogn.* **35**(8), 1785–1791 (2002).
11. P. Scheunders, "A genetic C-means clustering algorithm applied to color image quantization," *Pattern Recogn.* **30**(6), 859–866 (1997).
12. I. S. Hsieh and K. C. Fan, "An adaptive clustering algorithm for color quantization," *Pattern Recogn. Lett.* **21**, 337–346 (2000).
13. T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu, "An efficient K-means clustering algorithm: analysis and implementation," *IEEE Trans. Pattern Anal. Mach. Intell.* **24**(7), 881–892 (2002).
14. Y. L. Huang and R. F. Chang, "A fast finite state algorithm for generating RGB palettes of color quantized images," *J. Infor. Sci. Eng.* **20**, 771–782 (2004).
15. M. B. Al-Daoud and S. A. Roberts, "New methods for the initialization of clusters," *Pattern Recogn. Lett.* **17**, 451–455 (1996).
16. J. M. Peña, J. A. Lozano, and P. Larrañaga, "An empirical comparison of four initialization methods for the K-means algorithm," *Pattern Recogn. Lett.* **20**, 1027–1040 (1999).
17. S. C. Cheng and C. K. Yang, "A fast novel technique for color quantization using reduction of color space dimensionality," *Pattern Recogn. Lett.* **22**, 845–856 (2002).
18. Y. C. Hu and B. H. Su, "Accelerated K-means clustering algorithm for color image quantization," to appear in *Imaging Sci. J.*.
19. Y. C. Hu and B. H. Su, "Accelerated fast pixel mapping scheme for color image quantization," to appear in *Imaging Sci. J.*.
20. Y. Linde, A. Buzo, and R. Gray, "An algorithm for vector quantizer design," *IEEE Trans. Commun.* **28**, 84–95 (1980).
21. J. Foster, R. M. Gray, and M. O. Dunham, "Finite-state vector quantization of waveform coding," *IEEE Trans. Inf. Theory* **31**, 348–359 (1985).
22. International Commission on Illumination (CIE), *Industrial Colour-Difference Evaluation*. Publication CIE 116-95, Bureau Central de la CIE, Vienna (1995).
23. International Commission on Illumination (CIE), *Recommendations on Uniform Color Spaces, Color Difference Equations, Psychometric Color Terms*, Publication CIE 15 (E.-1.3.1), Supplement No.2, Bureau Central de la CIE, Vienna (1971).
24. X. Zhang and B. A. Wandell, "Color image fidelity metrics evaluated using image distortion maps," *Signal Process.* **70**(3), 201–214 (1998).

**Yu-Chen Hu** received his PhD degree in computer science and information engineering from the Department of Computer Science and Information Engineering, National Chung Cheng University, Chiayi, Taiwan, in 1999. Currently, Dr. Hu is a professor in the Department of Computer Science and Information Engineering, Providence University, Sha-Lu, Taiwan. He is a member of SPIE and IEEE. He is also a member of Phi Tau Phi Society of the Republic of China. His research interests include image and signal processing, image compression, information hiding, and data engineering.

**Ming-Gong Lee** received his MS in computer science & information management from Providence University in Taiwan in 2005. He currently works as an assistant engineer at Chunghwa Telecom. His research interests include image processing, image compression, and networking.