Adaptive Colormap Selection Algorithm for Motion Sequences

John L. Furlani
SunSoft, Inc.

Leonard McMillan
The University of North Carolina

Lee Westover
Division, Inc.

ABSTRACT: *We present a simple and intuitive algorithm for the quantization of full-color images which has been designed to apply to static images and motion sequences equally well. Our technique eliminates the perils of hardware colormap flashing which is inherent in other well known algorithms for selecting colormap representatives. We compare our technique with existing static image colormap generation techniques to show the quality of the resultant quantization.*

Introduction

Color frame buffer architectures which represent each displayed pixel as an index into a colormap are commonplace in the computer industry. While such architectures are suitable for many applications, they do not provide suitable fidelity for the errorless representation of arbitrary full-color continuous tone images. However, techniques have been developed which attempt to select an optimized colormap for a specific full-color image. These determine a set of colors which best represent the color gamut of the input image.

When an input image is mapped from its original description to a set of representatives it undergoes a quantization process. The differences between the original values and the resulting values are known as quantization errors. It is desirable to minimize quantization errors. An important step in the quantization process is the selection of an appropriate set of representatives, which for the application being discussed are the various colormap entries. In general, this selection process falls into one of two classes: *unbiased* selection where the choice of representatives is completely independent of the source being quantized, and *biased* selection where representatives are chosen to represent a specific source. A common unbiased representative selection scheme is "uniform" or "colorcube" quantization. In this scheme, each dimension of the color space is subdivided into a fixed number of levels. The total number of representatives is determined by the product of the number of subdivisions in each dimension.

Common variants include the unequal distribution of subdivision levels among the dimensions and variable spacing of these subdivisions based on some perceptual model. These hybrid quantization schemes are still in the unbiased class.

Unbiased techniques are known to introduce visible contouring when the number of representatives is small. As a result, this technique is frequently used in conjunction with dithering [Bayer~73] [Floyd~75] [Jarvis~76].

The predominant computer graphics work on biased representative selection schemes was presented by [Heckbert~82]. Heckbert describes and contrasts two different biased selection techniques: a popularity algorithm and a median cut algorithm. Heckbert also found images quantized using his biased selection techniques were of strikingly higher subjective quality than those produced by unbiased techniques. [Gervautz~90] has also proposed a biased selection technique based on octree quantization. And [Wu~1991] has proposed a biased selection technique based on variance minimization.

[Heckbert~82] also made reference to a third biased technique which he called "A Fixed Point Algorithm for Improving a Quantizer," based on work originally done by [Lloyd~57] and extended by [Gray~80]. This type of quantization technique is well known in the signal processing discipline and is often applied to vector quantities. One particularly useful technique was developed by Linde, Buzo, and Gray [Linde~80]. It is commonly referred to as the LBG algorithm. The algorithm that we present is based on these earlier works and falls into this class of fixed-point algorithms. We will elaborate on many of the algorithmic subtleties related to the particular application of LBG to colormap optimization, as well as present some quantitative analysis and comparisons as extensions to median cut.

It should be noted that the coupling of the biased representative selection process to a single source image has generally limited its usefulness for image sequences. The major difficulties are a result of the hardware implementation, which typically consists of a single video lookup table, and a single indexed frame buffer. Given such a configuration, when image N of the sequence is updated to image $N+1$, a phenomenon known as *colormap flashing* occurs. This flashing is a result of color discontinuities when for a short period of time, frame N is displayed with the colormap from frame $N+1$.

There are two brute force solutions to this problem. The first is to constrain both the colormap update and the image update to occur before the first pixel of image $N+1$ is scanned out of the framebuffer. This has the drawback of limiting the speed at which video can be displayed on a simple framebuffer and it requires special code in a playback application for every framebuffer on which the video sequence will be displayed. The second approach requires the double-buffering of the scan-out memory. This might be accomplished by allocating half the available colormap for the display of even frames and the other half for odd, thereby reducing the number of representatives by a factor of two.

In this paper, we will introduce a simple and intuitive technique for the generation of optimized colormaps from full-color images, based on the works of [Lloyd~57] and [Linde~80]. We will also present a new *adaptive* class for representative selection, where the selection process is guided by a predictive model. And finally, we will present extensions of the algorithm which allow image sequences to be displayed on common framebuffer architectures without flashing.

Prior Work

Both [Heckbert~82] and [Gervautz~90] used subjective evaluations in the analysis of their selection techniques. While such qualitative techniques - based on subjective perception - have their use, we have chosen to use quantitative measures to facilitate comparative analysis. [Heckbert~82] defined "optimal" quantization as a process which minimizes some error metric for a given image and a given number of representatives. In an effort to clarify this definition, we have chosen to describe the aforementioned case as "optimal with respect to an error metric." There are many possible candidate error metrics which have been suggested for use in computer graphics, including mean square error (MSE), which is defined by Equation 1, and mean absolute error (MAE), which is defined by Equation 2.

Equation 1:

$$MSE = \frac{1}{MN}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}(pixel'[m,n] - pixel[m,n])^2$$

Equation 2:

$$MAE = \frac{1}{MN}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}\sqrt{(pixel'[m,n] - pixel[m,n])^2}$$

In the case of the median cut method, a biased scheme, it has been suggested by [Lippman~89] that when the "*largest* cell is selected for subdivision... this insures that the peak error is minimized." Heckbert also suggests that other criteria might be used to select the appropriate box for further subdivision. He describes one other criterion which he suggests would tend to minimize the mean square error better than the median criterion. At best these biased selection techniques only indirectly address the issue of minimizing a specific error metric.

In contrast, the fixed point class of selection schemes attempts to directly minimize the chosen error criterion. While an optimal solution is not guaranteed, [Linde~80] has proven that the algorithm converges in a finite number of iterations to at least some local minimum.

It should be noted that quantization error is not necessarily a good measure of perceptual quality. An good example of this is demonstrated by the dithering process where the addition of low-level noise enhances the perceived image quality yet increases the quantization error. As better perceptual error models are developed, they can easily be adapted for use as LBG decision criteria.

The LBG algorithm [Linde~80] is an iterative algorithm, which starts from some initial guess and eventually converges to a local minimum. The algorithm can be divided into two distinct parts, an analysis phase followed by an adaptation phase. In the analysis phase, statistics are gathered about the distribution of the input training sequence. An error metric is chosen which designate how well the quantization matches the input training sequence. In the adaptation phase, these metrics are used to migrate the current quantization values toward another quantization of the input training sequence. If the resultant quantization error is within a distortion threshold, ε, the quantization is considered final.

We apply the LBG algorithm to the selection of color-map entries for the display of both static and dynamic computer images. The first application is the generation of high-quality colormaps from still images. In its application to still images, we augment the basic migration aspects of the LBG algorithm with the outright setting of colormap values to facilitate a more rapid convergence. In addition, we show how the iterative nature of the algorithm naturally extends to a dynamic input stream of video images.

Throughout the remainder of the paper, we will refer to the algorithm we are describing as the Adaptive Colormap Selection (ACS) algorithm.

Applying ACS for Static Image Colormap Generation

In this section, we describe how the algorithm works for generating a colormap from a static image. In the next section, we extend this static image algorithm and use it for a sequence of images. Like the LBG algorithm, we break down the ACS algorithm into two distinct parts, an analysis phase followed by an adaptation phase.

The analysis phase is a straightforward nearest color computation from each pixel in the image to one of the entries in the colormap.

```
foreach pixel in the input image {
  determine the colormap entry, $E_i$, nearest
      to the pixel
  Add pixel's contribution to the following
      statistics kept for $E_i$:
    $N_i$: number of pixels represented by $E_i$
    $M_i$: mean value of pixels represented by $E_i$
    $S_i$: variance of pixels represented by $E_i$
    $D_i$: mean error of pixels represented by $E_i$
}
```

The adaptation phase is a bit more complex and is different from the adaptation in a straight translation of the LBG algorithm.

```
foreach colormap entry $E_i${
  if($N_i$ > Minusage) {
    migrate the entry toward $M_i$;
  } else {
    if($N_i$ == 0) assign entry more useful color
    else      label entry as under-utilized
  }
}
```

The LBG algorithm does not specifically handle the case where a given representative is under-utilized. [Linde~80] suggests that in such cases an arbitrary vector [color] be assigned to the under-utilized entry. Before assigning an arbitrary color to an under-utilized entry, we use the image statistics to assign a color which attempts to improve the perceived quality of the quantized image. This accelerates convergence and helps to reduce contouring and other artifacts by breaking up entries which have a large error or large usage by using more entries in that area.

The following sections describe in more detail each step in the algorithm.

Analysis Phase

The analysis phase gathers statistics about the image and performs a nearest color computation of each pixel in the image to an entry in the current colormap. In fact, some colormap entries may be marked as "unusable" and thus not available to the nearest color calculation. What defines an "unusable" entry is discussed in detail in Section 3.2.1. After a pixel has been mapped to its nearest colormap entry, some statistics are gathered for the entry in the colormap. The statistical elements gathered are:

- The total number of pixels mapped to the entry.
- The total of each color component which is used for computing the average color mapped to the entry.
- The total of the square of each color component which is used for computing the standard deviation of the colors mapped to the entry.
- The total error caused by the nearest color mapping of the pixel to the colormap entry. The error is defined as the euclidean distance between the pixel value and the colormap entry value.

Adaptation Phase

Once the statistics are gathered for the entire image, the adaptation phase of the algorithm begins for generating the next iteration's colormap.

Entry Classification

Each of the entries in the colormap is placed into one of three classes based on the gathered statistics.

The first class of entries is made of those which are considered to have not been used enough during this step for continuing their use in the next step. Determining whether an entry should be used in the next step is based upon a number of factors.

A usage threshold is used for determining whether an entry was used "enough" times. This threshold is stochastically determined between high and low thresholds which are based on the number of pixels each colormap entry would represent given a uniform distribution of pixels to colormap entries. We found that tying the value threshold to the optimal distribution ensured it was independent of the size of the input image. We did not find an optimal single level for this threshold. Some input sequences produced better results with a lower threshold and others with a higher threshold. Determining the threshold stochastically produced better images than selecting a single threshold for the iteration. If an entry's count is below the threshold, it is classified as "under-utilized."

The number of pixels which use a particular colormap entry may not sufficiently describe the importance of the pixels which utilize the colormap entry. Often, an image will have a small region which contains a unique color. These colors are not used by many pixels, but if the color were to be dismissed from the image the image detail and quality would greatly suffer. To avoid removing these important colors from the colormap, before an entry is actually turned off there must be another colormap entry which has a similar color. We divide the colorspace into 64 uniform subregions. If a color entry is the only entry in its subregion, then it will not be "turned off." An entry which has been "turned off" cannot be used by the nearest color computation in the next iteration.

The second class of entries is made of those which are considered to have been used enough during this step for their continued use in the next step. The colormap value associated with these entries will migrate toward the average color of the pixels which were mapped to this entry during the nearest color computation. The color value migration process is described in the Entry Migration section.

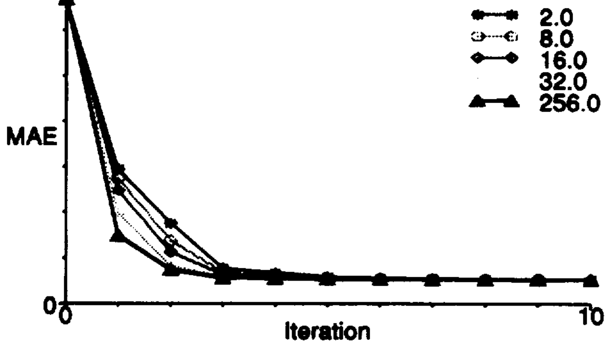
The third class of entries is made of those which were not used at all during this step and thus can be placed anywhere in the colorspace. These are the entries which were "turned off" during the previous iteration due to their underutilization. Since they were not used at all during this iteration, we are free to move them anywhere in the colorspace.

Entry Migration

Colormap entries which were classified as being "used enough" in the classification step are now migrated to the average of the colors which were mapped to the entry. The total of each color component is divided by the total number of times the entry was used by the image. This represents a desired color which the colormap entry should more closely match.

We introduce two parameters, t_{max} and α, which are used to control the migration of colors. The migration parameter, t_{max}, limits the euclidean distance a colormap entry can migrate in a single iteration. The error reduction parameter, α, controls the rate at which the total MSE for a colormap entry is adjusted after a migration. The total MSE for an entry is modified by α when its color is migrated.

FIGURE 1. How t_{max} affects the convergence rate.



When one is computing the colormap for a single image, t_{max} is usually set to a value which permits unlimited movement of entries throughout the colorspace. For still image colormap generation, colormap flashing is not an issue.

Resetting Entries

Colormap entries which were not used in the current pass can be set to any value in the colorspace. The primary goal is to reduce the total MSE generated by mapping the image to the colormap. In addition, an unacceptable artifact from using colormaps is contouring or banding because a single color is in the colormap for a region which contains many slightly different colors in the original image. To help alleviate this, we set the new colormap entries based on algorithms for reducing the MSE and the total number of pixels a single entry represents.

For each unused entry, we perform the following steps:

- Locate a target colormap entry with either the highest MSE or the highest usage count.
- Compute the standard deviation of the pixels mapped to the target colormap entry.
- Place the new entry within a radius equal to the standard deviation away from the target colormap entry.
- Reduce the MSE or the usage count of the target colormap entry by α.

The above steps have the effect of providing more colors in areas of the color space which are lacking colors as exemplified by a high MSE value. In addition, colors with too many pixels using them will have more colormap entries allocated to their area in hopes of bringing down the total number of pixels using a single color and thus reducing contouring.

If a target colormap entry has a high MSE or a high usage count, we only place a few new colormap entries using the aforementioned steps around it. Once we've placed six new colormap entries (one for each side of the cube surrounding the target colormap entry), we set its total error and usage count to zero. This ensures the colormap entry will not be chosen again for the assignment of other available colormap entries. If all of the used colormap entries have met these criteria, we assign the remaining unused colormap entries to random values. We have found through experimentation that placing colors randomly through the colorspace provides good results in this extreme case. In fact, this process provided the best results of the methods we attempted. The most common case for this to occur is when an image has a single color. Using random values has proven to be effective at accelerating convergence for images after a single-color image.

Cleanup and Return

Finally, we check through the colormap for any duplicate colormap entries. Any duplicates which are located are marked as "unusable" for the next iteration. Colormap entry duplication does not occur often, but it is not useful to allocate multiple entries to the same color.

The Final Colormap

Figure 2 graphically depicts the ACS algorithm. Initially the colormap is set to a 6x6x6 uniformly quantized colorcube. Figure 2 shows the resulting colormaps after each of the first four iterations. A sphere denotes the position of the color representatives prior to adaptation. The line segment originating from some spheres shows the color's migration path.

FIGURE 2. The First Four Iterations for Lenna Image

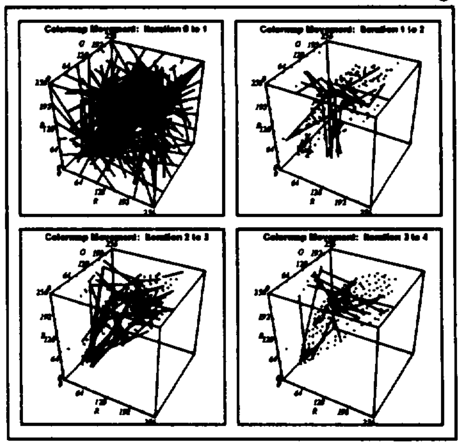


Figure 3 shows the resulting colormap after 5 iterations of the ACS algorithm. The resulting colormap generated by the median cut algorithm on the same image is shown in Figure 4 for comparison.

FIGURE 3. ACS Algorithm after Five Iterations for Lenna

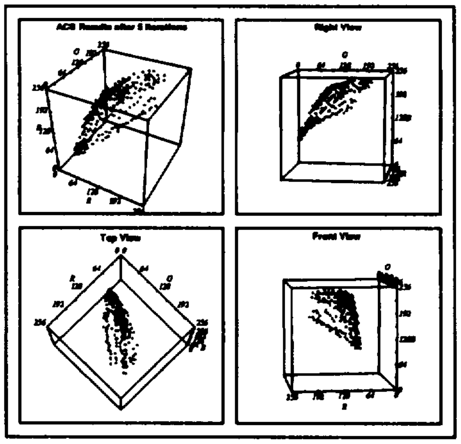
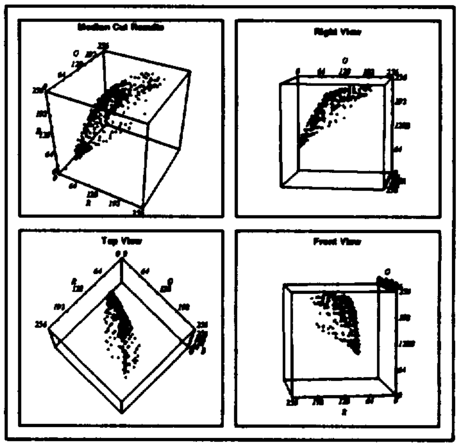


FIGURE 4. Final Median Cut Colormap for Lenna Image



Applying ACS to a Sequence of Images

In the previous section, we presented a simple and intuitive iterative algorithm which produces a high-quality colormap from a static image. However, the unique abilities of the ACS algorithm are exemplified when the algorithm is applied to a dynamic sequence of images. If any of the standard colormap generation algorithms described in section 2.0 were simply applied to sequential frames, colormap flashing problems would occur because the migration of colormap entries from one frame to the next are arbitrary.

The extended ACS algorithm is as follows. For each frame of a sequence, the analysis phase of ACS is performed and thus is simultaneously quantizing frame N with the current colormap. As mentioned earlier, the analysis phase of the algorithm performs nearly all of the operations required to quantize the N^{th} image. Following the quantization of frame N, the statistics gathered during the analysis phase are used to generate a colormap for the $N+1$ frame.

Within this framework it is possible to eliminate the colormap flashing problems of the other colormap generation algorithms by a simple reinterpretation of the migration parameter t_{max}. Through visual experimentation, one can adjust t_{max} to limit the amount each colormap entry can move between any two frames to below the just-noticeable-difference threshold. We have found that limiting the euclidean distance a colormap entry can travel, t_{max}, to between 9 and 12 gives good results.

What allows the ACS algorithm to work so well for video sequences is the inherent characteristics of traditional or "standard" video. In "standard" video, scene changes generally occur once every few seconds and rarely more than three or four times per second. After experimentation on a wide range of video sequences, we determined that a 2 to 3 frame margin of non-optimal colormaps is available before a viewer detects the colormap error. The modification we made to the basic LBG algorithm which provides an accelerated level of convergence is the process of resetting under-utilized colormap entries.

This modification assists the ACS algorithm in handling drastic changes in the color scheme between scenes. If a scene change is not very drastic, the migration step is the primary path for generating the $N+1$ colormap. But, if the change is drastic, the entry reset step provides a fast adaptation. In this case, a few colormap entries are used on the first frame of the new scene which provides a large number of under-utilized entries which can be reset to match the new color scheme.

When applying the ACS algorithm to a sequence of images, we limit the cases for which we perform the migration step. If the distance a pixel desires to move is less than $\sqrt{3}$, it is not moved at all. This restriction was added because we found that in a sequence of images, colors will tend to migrate often for little or no gain. This motion produces the unpleasing visual artifact of static scenes appearing to move and jitter. Limiting the movement nearly eliminates this artifact.

The ACS algorithm has been used for off-line video quantization and compression with real-time decompression and display. The ability to implement the ACS algorithm for real-time (30 fps) operation is limited by the nearest color search.

Results

Table 1 shows the resultant MAE calculated for several images using the median cut, the popularity, the variance minimization, and the ACS algorithms. The median cut technique maintained all 8 bits of each primary. The popularity technique was run maintaining both 4 and 5 significant bits. We have found that this prequantization step is necessary to achieve reasonable results from the popularity algorithm. The ACS algorithm was run for 16 iterations with $t_{max} = 256.0$ and an initial colormap of a 6x6x6 color cube. As shown, the ACS algorithm usually achieves a lower MAE than the median cut method after 4 to 8 passes, but, on a couple of images the variance minimization technique produces somewhat better results. What is important about this table is that the ACS technique consistently produces colormaps for static images which are nearly as good as or better than techniques designed solely for the generation of colormaps from static images. Color Plate 1 and Color Plate 2 are example static images generated using the ACS algorithm after 0, 1, 2, and 3 passes when starting with a grey-scale colormap. The lower left-hand corner of each image contains the colormap which corresponds to that image.

TABLE 1. MAE and Maximum Pixel Error for Various Images and Methods

Results												
	lenna		mandrill		lake		f16		tiffany		peppers	
	MAE	Max E	MAE	Max E	MAE	Max E	MAE	Max E	MAE	Max E	MAE	Max E
Median Cut	5.7	108.7	10.1	59.2	7.1	109.8	3.6	75.0	5.4	88.5	7.6	74.6
Popularity 5	8.1	107.2	22.2	126.8	15.5	130.3	4.9	78.0	7.6	179.6	11.9	94.6
Popularity 4	7.8	99.6	11.6	75.3	8.7	112.8	7.7	70.1	13.2	63.4	9.0	78.5
Variance Min	5.5	108.0	9.9	49.0	7.1	86.8	3.5	31.8	5.1	61.1	7.3	68.3
ACS 1 pass	7.9	52.9	12.5	43.3	9.5	37.8	8.5	27.9	10.9	31.9	9.7	42.2
ACS 2 passes	6.6	31.3	11.7	49.1	8.4	85.0	5.8	79.5	7.8	64.4	8.7	62.4
ACS 4 passes	5.7	45.4	10.3	53.0	7.3	95.5	3.6	45.1	6.3	63.7	7.9	73.0
ACS 8 passes	5.5	108.6	9.9	63.9	7.0	94.2	3.3	80.2	5.7	55.5	7.6	73.0
ACS 16 passes	5.4	108.6	9.7	58.4	6.9	94.2	3.3	66.5	5.3	65.2	7.5	72.4

In an effort to understand the ACS algorithm's sensitivity to the choice of an initial colormap, we studied four different starting conditions. We were particularly interested in the number of passes required for suitable convergence. The first colormap was the colormap generated by the median-cut algorithm. The second colormap was the 6x6x6 uniformly quantized colorcube. The third colormap was randomly generated. The fourth was a grey-scale colormap. Notice in Figure 5 and Figure 6, while the first few passes have different MAE, all four initial colormaps stabilized within 4 or 5 iterations.

FIGURE 5. Lenna Image: Sensitivity to Initial Colormap

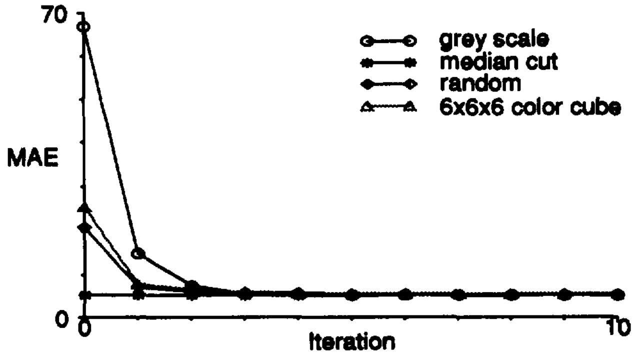
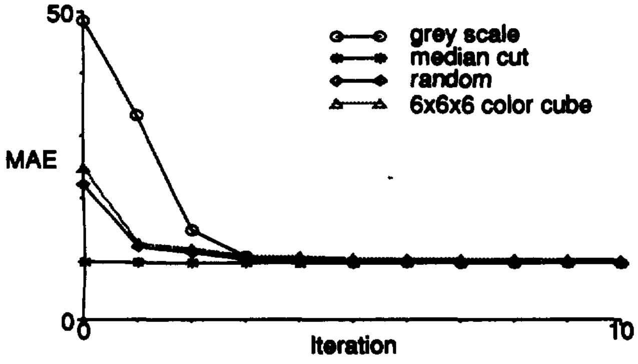


FIGURE 6. Mandrill Image: Sensitivity to Initial Colormap



The ACS algorithm was also used to generate colormaps for two Sun Microsystems commercials, "Target" and "Mifkin."

"Target" is a sequence of video with only moderate color scheme changes for much of the video. It exhibits how the ACS algorithm's migration step will keep the colormap for every frame at a near-optimal level even though the color scheme is changing slightly. For example, if the variance minimization technique is used to generate a colormap on the static image at frame 100 of the "Target" video, the resultant MAE is 7.32. The ACS algorithm generated a colormap with an MAE of 7.37 for the same frame.

"Mifkin" is a sequence of video with more dramatic color scheme changes between scenes. It shows how rapidly the ACS algorithm adapts to a new color scheme after a drastic scene change. Like the "Target" video, the ACS algorithm keeps the colormap at near-optimal levels for most scenes. For example, if the variance minimization technique is used to generate a colormap on the static image at frame 500 of the "Mifkin" video, the resultant MAE is 6.04. The ACS algorithm generated a colormap with an MAE of 5.73 for the same frame.

Figures 7 and Figure 8 show a plot of the MAE for each frame for the entire "Target" and "Mifkin" video sequences respectively. It is easy to detect a scene change in the "Mifkin" video and note how rapidly the colormap error is reduced.

FIGURE 7. MAE for Each Frame of "Target" Video

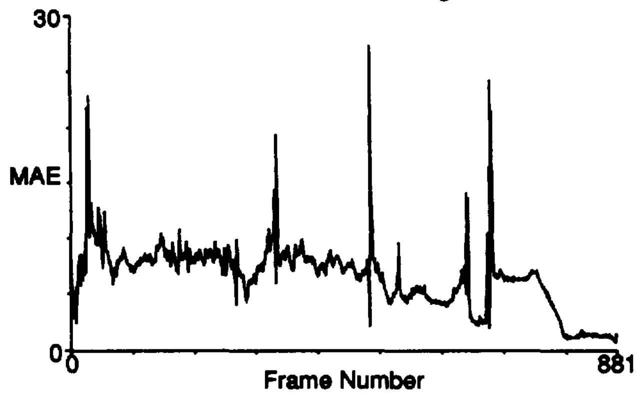
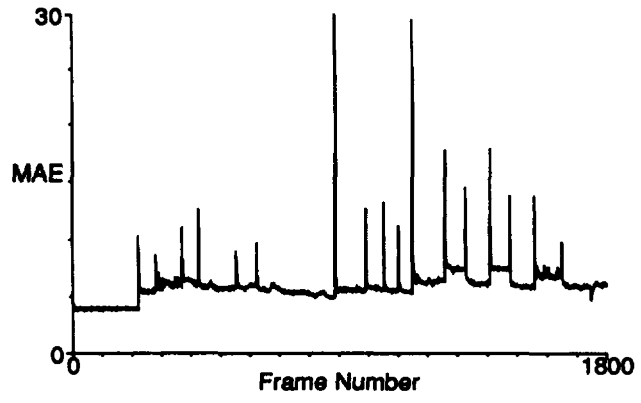


FIGURE 8. MAE for Each Frame of "Mifkin" Video



Color Plate 3 shows 20 sequential frames from the "Target" video sequence. Note the colormap in the lower left-hand corner of each frame. This sequence shows how the colormap changes through wildly varying color schemes. Color Plate 4 is 4 frames from the "Mifkin" video sequence which bridge a scene change with dramatically different color schemes. Although the static image quality of the frame immediately following the scene change is very poor, it contains enough color for the 1/30th of a second it is present on the screen to ensure the viewer's eyes do not notice the transition to the nearly-optimal frame two frames later.

Conclusions

We have presented a simple new two-phase algorithm for determining optimal colormaps for full-color images. For static images, the algorithm generates locally optimal colormaps in a small number of iterations. Since the migration behavior of individual colormap entries does not matter for static images, these maps can be generated using a maximal t_{max}. This allows the colormap to converge quickly. The first phase of the ACS algorithm is simply a search for the colormap entry that minimizes some error metric and the collection of relevant statistics. These statistics are used during the adaption phase to migrate colormap entries in order to reduce the error metric. The first phase reduces to a set of nearest neighbor searches that benefit from any nearest neighbor search optimization. Therefore, this method is a simple, intuitive and cost effective alternative to the median cut algorithm.

We generalized the iterative approach to be an adaptive approach for use with motion sequences. Instead of iteration on a single image, the algorithm uses the results for frame N as the seed for frame $N+1$. This algorithm is well suited for motion sequences for two main reasons. Since the algorithm is run for only one pass on any given frame and the vast majority of work is in the nearest neighbor search which is required of all biased algorithms, we have found that this algorithm runs significantly faster than alternative methods. The algorithm also allows one to trade off colormap flashing effects with convergence time by modifying t_{max}. We also showed that the ACS algorithm consistently produces colormaps for nearly every frame in a sequence of images as good as other colormap generation algorithms would on each static image. Finally, we discussed how proper selection of the t_{max} migration parameter eliminated the colormap flashing problems found when other colormap generation algorithms are used on motion sequences.

Author Information

John L. Furlani

John L. Furlani is currently a senior software engineer at SunSoft, Inc. as a member of the XIL Imaging and Video Library development team. In addition, he is currently a part-time Ph.D. student in the Computer Science Department at Duke University. His research interests include software design, video compression technologies, and nomadic computing. He received his B.S. of Electrical and Computer Engineering from the University of South Carolina at Columbia in 1990.

John can be reached via e-mail at j.furlani@ieee.org.

Leonard McMillan

Leonard McMillan is currently a Ph.D. student in the Computer Science Department of the University of North Carolina at Chapel Hill, on leave of absence from Sun Microsystems. His research interests include computer graphics, computer vision and image processing and compression. He received bachelor's and master's degrees in Electrical Engineering from Georgia Institute of Technology in 1983 and 1984.

Leonard can be reached at the Department of Computer Science, CB #3175 Sitterson Hall, Chapel Hill, NC 27599-3175, or via e-mail at mcmillan@cs.unc.edu.

Lee Westover

Lee Westover is currently a software architect for Division Inc., a virtual reality hardware and software company located in Chapel Hill, North Carolina. He received his B.S. of Computer Science from Michigan State University in 1983 and his M.S. and Ph.D. of Computer Science from the University of North Carolina at Chapel Hill in 1986 and 1991, respectively. His research interests include video and audio compression technologies and volume rendering techniques. In his spare time, he is trying to break the 3 digit barrier in his golf game.

Readers may contact Westover at Division Inc., The Courtyard, Number 10, 431 West Franklin Street, Chapel Hill, NC, 27516 or via e-mail at lee@divnc.com.

References

Bayer, B. E., [1973] "An Optimum Method for Two-Level Rendition of Continuous-Tone Pictures," *Conference Record of the International Conference on Communications*, pages 26.11-26.15, 1973.

Equitz, W. H., [1989] "A New Vector Quantization Clustering Algorithm," *IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. 37*, no. 10, pages 1568-1575, October 1989.

Floyd, R. and L. Steinberg, [1975] "An Adaptive Algorithm for Spatial Gray Scale," *Society for Information Display 1975 Symposium of Technical Papers*, page 36, 1975.

Gervautz, M, and W. Purgathofer, [1990] "A Simple Method for Color Quantization: Octree Quantization," *Graphics Gems*, Edited by A. S. Glassner, Academic Press, Inc., San Diego, CA, pages 287-293, 1990.

Gray, R. M., J. C. Kieffer, and Y. Linde, [1980] "Locally Optimal Block Quantizer Design," *Information and Control, vol. 45*, pages 178-198, 1980.

Heckbert, P., [1982] "Color Image Quantization for Frame Buffer Display," *Computer Graphics, vol. 16*, no. 3, pages 297-307, July 1982.

Jarvis, J. F., C. N. Judice, and W. H. Ninke, [1976] "A Survey of Techniques for the Display of Continuous Tone Pictures on Bilevel Displays," *Computer Graphics and Image Processing, vol. 5*, no. 1, pages 13-40, March 1976.

Linde, Y., A. Buzo and R. M. Gray, [1980] "An Algorithm for Vector Quantizer Design," *IEEE Transactions on Communications, vol. 28*, no. 1, pages 84-95, January 1980.

Lippman, A. and W. Butera, [1989] "Coding Image Sequences for Interactive Retrieval," *Communications of the ACM, vol. 32*, no. 7, pages 852-860, July 1989.

Lloyd, S. P., [1957] "Least squares quantization in PCM's", *Bell Telephone Labs Memo*, Murray Hill, New Jersey, 1957.

Wu, Xiaolin, [1991] "Efficient Statistical Computations for Optimal Color Quantization," *Graphics Gems II*, Edited by James Arvo, Academic Press, Inc., San Diego, CA, pages 126-132, 1991.