

Dynamic Color Quantization of Video Sequences

Evgeny Roytman and Craig Gotsman

Abstract—We present an efficient algorithm for dynamic adaptive color quantization of 24-bit image (video) sequences, important in multimedia applications. Besides producing high-fidelity 8-bit imagery, our algorithm runs with minimal computational cost and the generated colormaps are robust to small differences in consecutive images. Apart from the two standard color quantization tasks, colormap design and quantizer mapping, our algorithm includes colormap filling—an operation unique to dynamic color quantization. This task solves the problem of screen flicker, a serious problem in dynamic quantization of image sequences, resulting from rapid changes in display of colormaps. Our solution is based on two ideas: including in the current colormap a small set of color representatives from the previous image; assigning representatives to the colormap entries in an order that reduces the difference between contents of equal entries in consecutive colormaps. Our algorithm runs in near real time on medium-range workstations.

Index Items—Color quantization, colormap, video.

I. INTRODUCTION

VIDEO monitors display color images by modulating the intensity of three primary colors (red, green, blue) for each pixel of the image. In a digital color image each primary color is usually quantized to 8 bits of resolution. Thus, full-color digital display systems use 24 bits to specify the color of each pixel on the screen.

In many applications the cost of high-speed video memory needed to support such a full-color display on a high-resolution monitor is not justified. An alternative, used by many currently available displays, is to provide a limited number of bits, such as 8, for specifying the color of each pixel. Each of these $2^8 = 256$ values is then used as an index into a user-defined table of colors. An entry in the table contains a 24 bit value which specifies the colors red, green and blue components. In this way, the user is allowed to select a small subset of colors, called a colormap (or palette), from the full range of $2^{24} = 16,777,216$ colors.

Since natural images typically contain a large number of distinguishable colors (much more than 256), displaying such images with a limited colormap is difficult. The process of selecting a small number of representative colors for an image of higher color resolution is called *color quantization*. The visual quality of images quantized to a smaller number of bits depends critically on how this limited set of colors is chosen. The theoretical base for color quantization is *vector quantization* [1], [2].

The typical approach to color quantization involves algorithms for performing two tasks. Given an image, the first task,

called *colormap design*, selects the best possible set of representative colors for the image. These colors comprise the image colormap. The second task, called *quantizer mapping*, maps each color in the image to a color of the colormap.

There are two general classes of quantization methods: *fixed* and *adaptive*. In fixed quantization, a predefined set of display colors and a fixed mapping from image colors to display colors are used. Both of these do not depend on the input image. *Uniform* quantization is a commonly used fixed quantization method. In adaptive quantization, the choice of the colormap depends on the statistical distribution of the colors in the input image. There are several algorithms for adaptive quantization, of which the most widely known are: *Popularity* algorithm [3], *Median-Cut* algorithm [3], *Variance-based* algorithm [4] and *Octree* algorithm [5].

Fixed quantization is easier to implement and runs faster than adaptive quantization, but the quality of images generated by the latter is significantly better. *Contouring* is a serious defect present in images generated by fixed quantization. This results because the color distribution of natural images varies widely, so many of the colors in the colormap are not used in the final image; they are wasted. By adapting a colormap to the color contents of the original image, we are assured of using every color in the colormap, thereby reproducing the original image more closely. In this paper we consider only adaptive methods of color quantization.

Until recently, most color quantization researchers and practitioners focused on the processing of single images. However, the advent of multimedia and low cost 24-bit video digitizers require algorithms for color quantization of image sequences. Three different approaches to the quantization of image sequences are possible:

- *Fixed quantization*. Each image is quantized independently of the others using uniform quantization and then possibly dithered to distribute the quantization error. The colormap and pseudorandom dither matrix do not depend on the distribution of colors in the original image. Being extremely simple, this algorithm is built into the hardware of some graphics workstations. It runs quickly but the results are not visually pleasing.
- *Static adaptive quantization*. One “optimal” colormap is generated for the entire sequence of images. This colormap is constructed in a preprocessing stage and its contents depend on the color distribution of the *entire* sequence. This is applicable only for short sequences, and requires the entire sequence to be available prior to display. It is obviously unsuitable for live video applications (e.g., teleconferencing).
- *Dynamic adaptive quantization*. Each image is quantized independently, producing a different colormap for each

The authors are with the Department of Computer Science, Technion-Israel Institute of Technology, Haifa 32000, Israel; e-mail: evgen@cs.technion.ac.il and gotsman@cs.technion.ac.il.

IEEECS Log Number V95022.

image of the sequence. This method can potentially achieve the best display quality. In practice it has two serious drawbacks. Colormap design and quantizer mapping for each image in the sequence are time consuming, and may be prohibitive for real time applications. Moreover, frequent changing of colormaps leads to screen flicker—high-frequency noise during sequence display.

In this paper we present an algorithm for dynamic adaptive color quantization of image sequences.¹ Besides the two standard color quantization tasks—colormap design and quantizer mapping, this algorithm includes a new component, called *colormap filling*. This assigns select color representatives to the colormap entries in a way which reduces screen flicker during change of the display colormap. In addition, the solutions for each of these tasks meets very strict computational time requirements. All our algorithms have been implemented and tested on Sun and SGI workstations, and achieve near real-time performance.

This paper deals with vector quantization in the RGB color space. As will be stated in the following section, the distance between two colors is quantified as the Euclidean distance between these points in the 3D RGB cube. This implies that to the human eye, the visual difference between colors located at the vertices of an equilateral triangle in RGB space should be the same. This is known to be false. For this to be true, a so-called “perceptual” color space must be used. The LUV color space is widely considered to be perceptual, but the transformation of point sets between the RGB and LUV color spaces is highly non-linear (see eg., [17] for a survey of the various color spaces, and conversions between them). Theoretically, it would be better to perform quantization in LUV space, as the distribution of colors in natural images tends to have some distinctive patterns in this space. However, since the main focus of this paper is rapid image *sequence* quantization for (near) real-time applications, and most video digitizers output RGB data, the additional overhead incurred by conversion to and from LUV space is prohibitive. Nonetheless, the problem of flickering, due to colormap changes, exists for quantization in all color spaces, and the techniques presented in this paper for the minimization of flickering are applicable to colormap construction in any color space.

The rest of this paper is organized as follows: In the next section colormap design, colormap filling and quantizer mapping - the three tasks of the dynamic adaptive quantization algorithm - are introduced. In Section III we review existing adaptive color quantization algorithms for single images. Section IV contains a description of our dynamic adaptive color quantization algorithm for image sequences. The algorithmic solutions for each of the three tasks are described. Experimental results and observations on the behavior of the algorithm are given in Section V. We conclude in Section VI.

A preliminary version of this paper was presented at Computer Graphics International '94 [16].

1. After this paper was submitted for publication, we became aware of the work of Furlani et al. [15], who present a different solution for this problem.

II. PROBLEM FORMULATION

In this section we describe the three components of a dynamic adaptive color quantization algorithm for image sequences. First some notation and definitions.

NOTATION 1. $RGB = \{(r, g, b) \mid 0 \leq r, g, b < 2^8, r, g, b \text{ integers}\}$ is the discrete 24 bit three dimensional RGB color space.

RGB is the set of all 256^3 colors present in a 24-bit color system.

NOTATION 2. $d(s, r)$ is the Euclidean distance between two points $s, r \in RGB$.

DEFINITION 1. $S = \{s_1, s_2, \dots, s_n\}$ is an **n-partition** of a discrete set P if:

- 1) $s_i \neq \emptyset$, for all $1 \leq i \leq n$.
- 2) $s_i \cap s_j = \emptyset$, for all $1 \leq i < j \leq n$.
- 3) $\bigcup_{1 \leq i \leq n} s_i = P$.

Elements of S are called **clusters** of the partition. A **partial partition** is one satisfying only conditions 1-2.

In the sequel we will be partitioning discrete sets of RGB colors which will either be the set of all colors in RGB space or the smaller set of RGB pixel values present in specific color images. In the former case, we will call the clusters of the partition **RGB cells**. In the latter case, we allow a cluster to contain multiple elements corresponding to more than one pixel having the same color.

DEFINITION 2. Let s be a cluster. The **centroid** of s is $cent(s) = \left[\frac{1}{|s|} \sum_{x \in s} x \right]$ (algebraic vector sum).

DEFINITION 3. Let $R = \{r_1, r_2, \dots, r_n\}$ be an n -partition of RGB and $S = \{s_1, s_2, \dots, s_n\}$ be an n -partition of the RGB colors of an image I . We say that R **induces** S and S is **induced by** R if for all $1 \leq i \leq n$, $s_i = I \cap r_i$. In this case, the cluster s_i is induced by cell r_i .

When a set of RGB color points is quantized to a smaller set of representatives, these representatives are listed in an indexed *color table*. A representative is then specified by a small number of bits through its index.

DEFINITION 4. Let $C = (c_1, c_2, \dots, c_n)$ be an RGB color table of n colors. A **quantizer mapping** is a function $Q: RGB \rightarrow C$. Through this mapping each 24-bit color is mapped to a $\lceil \log_2 n \rceil$ -bit integer that is an index into C . The pair $\mathcal{M} = (C, Q)$ is called a **quantizer**.

Once a partition has been defined on the colors present in an image, this naturally induces a simple quantizer on that image. The induced quantizer may then map all image colors contained in a cluster to the centroid of all those colors. If a partition P has been defined on RGB, this also naturally induces a quantizer on a given image, through the partition Q of the image pixel colors induced by P .

DEFINITION 5. Let $S = \{s_1, s_2, \dots, s_n\}$ be an n -partition of the colors of an image I . The **quantizer induced by** S is $\mathcal{M}_S =$

($cent(S), Q$), where $cent(S) = (cent(s_1), cent(s_2), \dots, cent(s_n))$ and Q is a centroid quantizer mapping, i.e., it maps each color $x \in s_i$ to $cent(s_i)$. (The mapping of colors not present in I is unspecified, therefore a quantizer induced by S is not necessarily unique.)

The quality of a given quantizer operating on a given image may be quantified by the Euclidean distances between the image colors and their representatives:

DEFINITION 6. Let $\mathcal{M} = (C, Q)$ be a color quantizer and I an RGB image of size $N_x \times N_y$. The **quantization root mean square error (QRMSE)** of \mathcal{M} on I is:

$$QRMSE(\mathcal{M}, I) = \left[\frac{1}{N_x N_y} \sum_{i=1}^{N_x} \sum_{j=1}^{N_y} d(I_{i,j}, Q(I_{i,j}))^2 \right]^{\frac{1}{2}} \quad (1)$$

NOTATION 3. Let I be an image and S an n -partition of I . When it is obvious which I is intended, $QRMSE(S)$ is short for $QRMSE(\mathcal{M}_S, I)$.

Most images contain many instances of the same colors, which can be grouped together for processing. The distribution of the distinct colors present in an image may be organized in a histogram, which is the standard input to a quantization algorithm.

DEFINITION 7. Let I be an image. A **histogram** of I is a 3D matrix $\pi_I = \{n(r, g, b) | 0 \leq r, g, b < 255\}$, where $h(r, g, b)$ is the number of occurrences of (r, g, b) in I . Each element (r, g, b) of the histogram is called a **bucket**.

A. Colormap Design

The first task of color quantization to n colors—*colormap design*—may be formulated as a large scale clustering problem. Given an image I , our goal is to find a n -partition $S = \{s_1, s_2, \dots, s_n\}$ of the colors in I , minimizing $QRMSE(S)$. It is known that the problem of finding a global minimum of $QRMSE(S)$ is NP-complete [6], [7]. Consequently, any computationally efficient solution to color quantization will be suboptimal. Existing approximation techniques may be divided into two categories: iterative optimization and heuristic methods. The iterative algorithm is the generalized Lloyd clustering algorithm proposed by Linde et al. [8] (also known as the LBG algorithm). It finds a local minimum of $QRMSE(S)$. This algorithm is computationally intensive, so unpractical for color quantization under time constraints.

Instead of finding local minima, heuristic methods are designed to produce an acceptable solution very rapidly. The basic heuristic method is divisive, iteratively subdividing RGB space into cells, inducing an n -partition on the image data. For simplicity, in most divisive algorithms the partitioning planes are parallel to one of the RGB coordinate planes. The different algorithms use different criteria to determine which cell to partition further, and where exactly to position the partitioning plane. At the end of this procedure, each of the partition cells is actually a 3D box. The centroids of the resulting n image clusters in these boxes are included in the image colormap. These colors are also called the image *representatives*.

B. Quantizer Mapping

Suppose that C is a given RGB colormap and I is an RGB image. From (1) it follows that the quantizer mapping Q of $\mathcal{M} = (C, Q)$, minimizing $QRMSE(\mathcal{M}, I)$ for a given C , must be a nearest-representative mapping. Computationally, computing the nearest representative is a very expensive operation. Even the two methods considered the most practical for multidimensional nearest-neighbor search, namely, use of k -d trees [9] and locally sorted search [3], are still not very efficient [10].

A faster, but suboptimal, alternative to nearest-representative mapping is to use the n -partition S obtained during the colormap design process and map the image colors to the representative, which is the centroid of the cluster containing the color. This is called *centroid mapping*. The value of $QRMSE$ produced by this quantizer is precisely $QRMSE(S)$.

C. Colormap Filling

Colormap filling takes place between colormap design and quantizer mapping in the quantization of a sequence of images. Since dynamic adaptive color quantization prescribes that a new colormap be produced for every image, an entry of the colormap may contain very distinct colors for consecutive images. As a result, when the sequence is displayed on the colormapped display, screen flicker between successive visualization of the images becomes a serious visual artifact. This phenomenon is caused by the fact that during a very short period of time, while the frame buffer of the display contains an image, but a new colormap (tailored to the next image in the sequence) is already active, there is a sudden change of the image colors until the next image is loaded. This is sharp and chaotic, unpleasant to the human eye. Fig. 1 demonstrates the effect of screen flicker during colormap switch between display of two consecutive images of the 'Table-tennis' sequence, quantized by the popular median-cut algorithm.

A simple solution to this problem is to divide the colormap to two parts and use different halves for color representatives of consecutive images. However this solution immediately reduces color resolution by a factor of 2. In special purpose display systems the screen flicker problem can be solved in hardware. A colormap switch and image loading may be done simultaneously by one atomic command during the longest "black" interval of the screen scan. However for most graphics systems, which run on a variety of hardware and are based on an asynchronous server-client communication model (such as X-windows), a software solution to this problem is necessary.

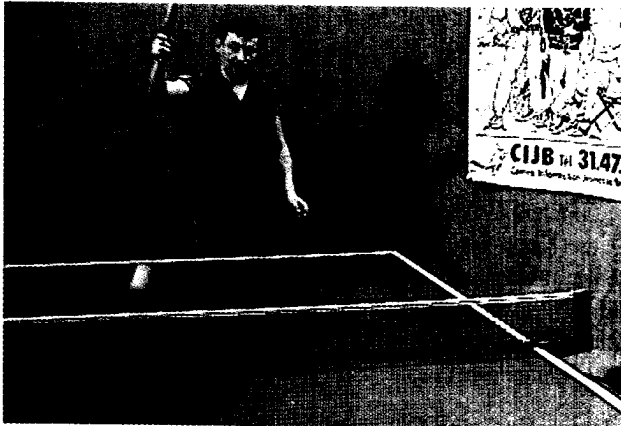
DEFINITION 8. Let I be an image and $C_1 = (q_1, q_2, \dots, q_n)$, $C_2 = (r_1, r_2, \dots, r_n)$ be colormaps. The **weighted distance** between C_1 and C_2 relative to I is:

$$D_I(C_1, C_2) = \frac{1}{N_x N_y} \sum_{i=1}^n W_I(q_i) \cdot d(q_i, r_i)$$

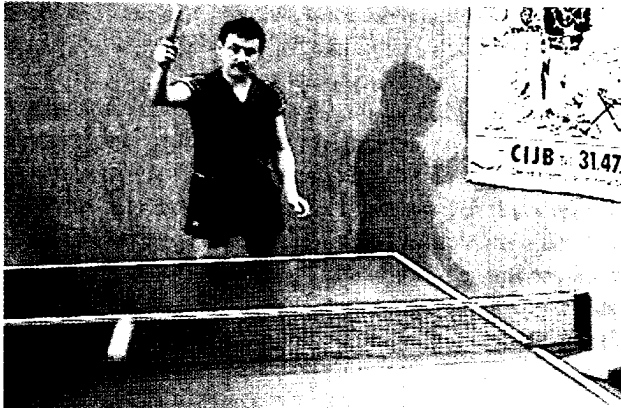
where $W_I(q_i)$ is the number of pixels in I mapped to color q_i (the weight of q_i in I).

The intensity of the screen flicker when changing active colormaps between I_1 and I_2 with colormaps C_1, C_2 is essen-

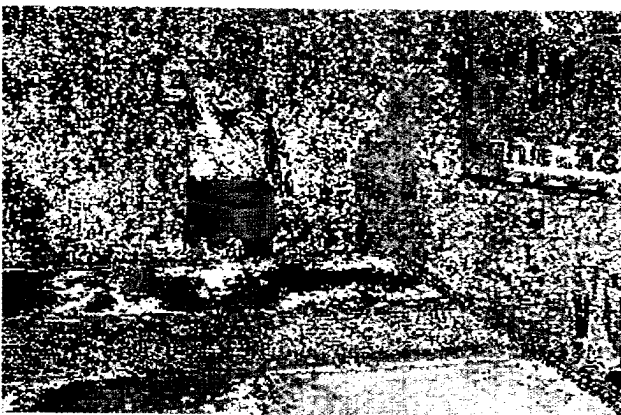
tially proportional to the value of $D_I(C_1, C_2)$. During colormap filling we try to solve the screen flicker problem by minimizing the value of $D_I(C_1, C_2)$. Our solution is based on two ideas: including in the image colormap a small set of representatives of the previous image; ordering the representatives in the colormap such that the difference between contents of the same entries in two consecutive colormaps is reduced.



(a)



(b)



(c)

Fig. 1. The screen flicker problem: (a), (b) Two consecutive images A, B in the 'Tennis' sequence quantized by the median-cut algorithm. (c) Screen flicker as it appears during colormap switch between display of the images (A displayed with colormap of B).

III. PREVIOUS WORK

A. Nearest-Neighbor Search

Given a fixed set of n representative colors in RGB , and an arbitrary color, finding the nearest-neighbor of the color among the n representatives is a basic part of optimal quantizer mapping. The brute force method is exhaustive search over all the color representatives, obviously very time consuming. Much time is wasted considering distant points which could not possibly be the nearest-neighbor. Below, two algorithms to expedite the nearest-neighbor search are surveyed.

A.1. k - d Trees

The classic k - d tree method for nearest-neighbor search by orthogonal partitioning was proposed by Friedman [9]. Let n be the number of given data points, distributed in k -dimensional space. The *root* of the tree represents a box containing all n points. Each non-terminal node has two sons. These sons represent two sub-boxes defined by the partitioning. The terminal nodes represent mutually exclusive boxes, which form a partition of the data space inducing a partition on the given n point data set. At each non-terminal node a partition is made orthogonally to the coordinate axis with the largest variance of the node data and passes through the median of the data distribution along this axis.

The k - d tree data structure provides a mechanism for examining only those points closest to the query point, considerably reducing the computation required to find its nearest-neighbor. The search algorithm is a recursive procedure. The argument to the procedure is the node under investigation. At each invocation the procedure chooses the son of its argument that contains the query point. When the search reaches a terminal node, the data in the terminal node is examined exhaustively to find a current nearest point R and the distance d_R from the query point to R . When control returns, the algorithm compares the value of d_R to d_p , the distance of the query point to the partition plane. If $d_R \leq d_p$, then none of the points on the opposite side of the partition can be nearer to the query point than R . If $d_R > d_p$, then the data of the opposite side of the partition must be searched (recursively).

For color quantization, the $3d$ tree requires $O(n)$ space and $O(n \log n)$ construction time. The expected time required to answer a nearest-neighbor query is $O(\log n)$. This algorithm performs well for a large number n of uniformly distributed data. Unfortunately, there are no reports about behavior of this algorithm in color quantization where n is small ($n \leq 256$) and distribution of color representatives in color space is not necessarily uniform.

A.2. Locally Sorted Search

The *locally sorted search* technique was proposed by Heckbert [3]. This technique is popular in color quantization practice.

Let n be the number of given data points, distributed in RGB . The color space is divided into $p \times p \times p$ cubical cells each containing a sorted list of data points. Each list includes all data points which may be the nearest-neighbor of some

point in that cell. Each list entry is a record with two fields: a data point ID and its distance from the nearest color in the cell. This distance is defined to be zero for data points inside the cell. Note that a given data point can participate in several lists.

To limit the length of the lists we eliminate data points which could not possibly be the nearest neighbors of any point inside the cell. To do this we find the data point nearest the center of the cell and compute its distance from the farthest corner of the cell. This an upper bound on the distance between any point in the cell and (the point's) nearest-neighbor. All data points whose distance from the cell is greater than this bound can be eliminated from the list, thereby accelerating the list sorting operation and conserving memory.

To compute the nearest-neighbor for query point x we first find the cell which contains x and then exhaustively search the list associated with that cell.

The storage requirement for this data structure is $O(p^3l)$, where p^3 is the number of cells and l is the average length of a cell's list. Computing distances from each data point to a cell takes $O(n)$ time, and the list sort takes $O(l \log l)$, so the preprocessing time is $O(p^3 n + p^3 l \log l)$. In practice the cells are computed dynamically on demand (when first needed). The query time using this data structure is $O(l)$.

In choosing p there is a trade-off between query and preprocessing costs. Many cells will lead to short cell lists and hence short search time, but high preprocessing time. With small p the long lists will result in more search time, but the preprocessing cost will be low. The optimal value of p depends on the number and distribution of queries.

B. Iterative Color Quantization Methods

B.1. The LBG Algorithm

The LBG algorithm [8] is a method used in vector quantization for iteratively refining a codebook [1]. This algorithm is a generalization of an algorithm for the design of scalar quantizers, developed by Lloyd in the mid 1950s. The LBG algorithm is quite general, and has also been applied to a wide variety of problems in pattern recognition and clustering [11]. Since the problem of designing an optimal codebook is equivalent to finding a set of palette colors, $\{c_1 \dots c_n\}$, which best approximate a set of given image colors, $\{x_1 \dots x_m\}$, the LBG algorithm is directly applicable to the color quantization problem. The algorithm can be formulated as follows:

- 1) **Choose an initial set of color representatives** $C = \{c_1 \dots c_n\}$.
- 2) **Partition the image into** $S = \{s_1, \dots, s_n\}$ **by the following rule:** $s_i = \{x: d(x, c_i) \leq d(x, c_j): 1 \leq j \leq n\}$
 $c_i \leftarrow \text{cent}(s_i) \quad i = 1, \dots, n.$
- 3) **Compute** $QRMSE(S)$. **If it has not changed by a pre-specified tolerance since the last iteration, stop. Otherwise return to step 2.**

The algorithm converges because for a given set of representatives it forms an optimal partition, and for a given parti-

tion it forms an optimal set of representatives (centroids).

As it is, essentially, a gradient-descent optimization method, this algorithm converges only to a local minimum of the quantization error. Despite this, the results of this algorithm have been found to be very effective [1]. Since a quantization error function will generally have many local minima and some can be much better than others, it is clear that the choice of the initial set of representatives plays a key role in determining the local minimum to which the algorithm converges. The disadvantage of the LBG algorithm is its computational cost. If n is the number of representatives, and m the number of image colors, the formation of clusters in step 2 requires nm distance calculations and this step must be performed at each iteration. Due to this cost, the LBG algorithm is not applicable for color quantization of images, where large numbers of pixels are involved.

Since the LBG algorithm can only improve (or at worst leave unchanged) the error of any given initial quantizer, it is best used as a post-processing algorithm for improving initial palettes produced by other techniques. Experimental results indicate that the LBG algorithm will improve initial color palettes of poorly quantized images, but it will not significantly change palettes of well-quantized images [2].

C. Heuristic Color Quantization Methods

In this section we describe color quantization algorithms used for adaptive quantization of single images. They do not obtain even a local minimum of $QRMSE$, but in practice achieve a small value of this quantity.

The input to all described algorithms is the image histogram. To simplify computations, during histogram generation, the color resolution of the processed image may be reduced to 5 bits per each red, green, blue color component (instead of 8 bits). This reduction of color resolution has no noticeable impact on the image quality, and the size of the histogram is reduced from 256^3 to 32^3 buckets.

In this section m denotes the number of buckets in the image histogram. As before, n denotes the size of the colormap.

C.1. The Popularity Algorithm

The *popularity* algorithm was the first algorithm proposed for adaptive color quantization of images. This algorithm is based on the assumption that the colormap can be constructed by finding the densest regions of the image color distribution. The popularity algorithm simply selects the n colors of the image with the highest frequencies and uses these colors for the colormap. It takes time $O(nm)$. This algorithm produces good results for many images, but performs poorly on ones with a wide range of colors. It often neglects colors in sparse regions of the color space.

Braudaway proposed an *improved popularity* algorithm [12]. In order to prevent the concentration of too many representatives in the neighborhood of one distribution peak, the frequency count of its neighboring colors is reduced after the selection of a representative color. This leads to a more uniform distribution of the representatives in the image histogram.

C.2. The Median-Cut Algorithm

The *median-cut* algorithm was proposed by Heckbert [3]. The idea behind this algorithm is that each of the colors in the colormap should represent an approximately equal number of pixels in the original image. The data structure used by the median-cut algorithm is a *3d tree* [9].

The *RGB* space is iteratively divided into rectangular boxes until n boxes are generated. At each step the box with the largest number of image pixels is divided into two subboxes, each containing an equal number of pixels. The orientation of the partition plane is chosen to be perpendicular to the coordinate axis with the largest variance of the image pixels within the box, and it passes through the median point of the coloredistribution projected along this axis. As a result of this operation, each final box contains approximately the same number of image pixels. The centroids of clusters inside these boxes are chosen to be representative colors in the image colormap. Generating the colormap will take $O(m \log n)$ time and $O(m)$ space. To improve the generated colormap, Heckbert proposes applying the iterative LBG algorithm [8]. After the colormap is generated, each color of the image is mapped to its nearest representative. To optimize nearest-neighbor search, Heckbert uses the locally sorted search technique.

C.3. Variance-Minimization Color Quantization

The *variance-minimization* algorithm for color imagequantization was proposed by Wan et al. [13]. The basic idea behind this algorithm is to adaptively assign more clusters to the regions with large quantization errors (large data variance) such that the total quantization error of the partition is reduced. As a result, in contrast to the median-cut method, different boxes in the final partition may contain quite different numbers of data points, but the contribution of each box to the quantization error is approximately equal.

NOTATION 1. Let A be a set of real numbers. $\sigma(A)$ denotes the variance of A .

DEFINITION 1. Let $A = \{a_1, a_2, \dots, a_n\}$ be set of real numbers. Point t is an **optimal cut-point** of A if it minimizes the function $P_A(x) = \sigma(A_1(x)) + \sigma(A_2(x))$, where

$$A_1(x) = \{a_i | a_i \in A, a_i \leq x\}, A_2(x) = A \setminus A_1(x).$$

Similar to the median-cut method, the variance-minimization algorithm iteratively divides image data into rectangular subboxes until n clusters are generated. However, its partition strategy is based on the minimization of the sum of variances of the data in the resulting subboxes. At each step the box with the highest variance of data is chosen for division. All data points in this box are projected onto each of the coordinate axes. For each projected distribution, the optimal cut-point is computed. The partition plane is then chosen perpendicular to the axis along which the reduction of expected variance is the largest among all projected distributions. The partition plane passes through the optimal cut-point of this axis.

After the required number of clusters is generated, their centroids are chosen to be representatives in the colormap. Quantizer mapping is done by mapping each image color to its nearest representative. The local search technique provides an

efficient way to perform this mapping operation.

The computational complexity of variance-minimization algorithm is $O(m)$ space and $O(nm)$ time. The variance-minimization algorithm produces much smaller quantization error and hence better image quality than the median-cut algorithm [13].

C.4. Octree Quantization

The octree quantization method was proposed by Gervautz and Purgathofer [5], where a regular subdivision of a cube into eight octants is used. Conceptually, starting with the entire *RGB* cube, subdividing cubes containing more than one image color, while retaining only cubes which do contain image colors, yields a tree whose terminal nodes are individual colors.

The tree is then *reduced* by an averaging process. Terminal nodes are replaced by their father node containing an average of the color in the terminal nodes. This reduction process continues until the number of terminal nodes is exactly n .

In practice, it is not necessary to build the complete octree. Instead, the building and reduction take place in one pass through the image data. The first n different pixels form the terminal nodes of a tree. A reduction is then performed, either on the deepest node, or on the node containing the fewest pixels. More image pixels are then added as terminal nodes until n terminal nodes are obtained, and the process repeated. Due to this, the complexity of the algorithm is $O(n)$ space and $O(m)$ time. Quantizer mapping to cluster centroids is by a simple search of the final octree.

The results of octree quantization depends on the order in which the image pixels are scanned. However, in addition to its advantages of speed and size, octree quantization has the effect of preserving contrasts in the image. Visually, there is little difference in final image quality between median cut and octree quantization.

C.5. Optimal Principal Multilevel Quantization

All color quantization techniques described above (except the simple popularity algorithm) share a common algorithmic trend—iterative orthogonal division of color space. However, realistic color image data are generally not distributed orthogonally in the color space. It is clear that in order to minimize the variance when splitting a cluster s into two clusters s_1, s_2 , the cutting planes should be taken normal to the principal axis along which the image data has a maximum variance in s rather than normal to one of the three coordinate axes. Although an algorithm combining iterative division techniques with cutting planes perpendicular to the principal axis of the image data can improve orthogonal cutting methods, it still achieves only a local minimum of *QRMSE*.

The algorithm proposed by Wu [14] permits optimization of a multilevel partition. It constructs an n -partition orthogonally to the principal direction of the image data. Wu's method is based on the following observation: Let I be a color image, and let $\{H_i; 1 \leq i \leq n\}$ be the sequence of cutting planes generated by a greedy partition of the image I . Then, for some λ , the principal axis of the resulting subsets created by the first few cuts $H_i, 1 \leq i \leq \lambda$, remains approximately the same as the principal axis of the original image data. Thus the planes $H_i, 1$

$\leq i \leq \lambda$ are almost parallel to each other and all approximately normal to the principal axis of I . This is due to the fact that the color distribution of natural images is not isotropic in the 3D RGB space. The variance of the image colors is usually larger in the intensity direction (Y component in YIQ color space, V in HSV space) than in the other two chromaticity dimensions. Consequently, a color quantizer can be improved by simultaneously choosing optimal multiple cuts H_i , $1 \leq i \leq \lambda$ against the principle axis of the image, rather than choosing H_i greedily and one at a time. The proposed dynamic-programming algorithm makes this non-greedy optimization computationally feasible. It finds H_i , $1 \leq i \leq \lambda$ in $O(\lambda m + N)$ computational time, where N is a number of pixels in the image. The space complexity of the dynamic-programming algorithm is $O(\lambda m)$. The algorithm terminates when none of the produced clusters has an orientation strongly biased in the principal direction of I . The value of λ varies from four to eight depending on the input image. Further division of the image is done by a standard iterative scheme. The centroids of the final clusters are used as color representatives of the image.

Due to its better adaptability to the color statistics of input images and its ability to minimize quantization error over multiple quantizer cells as a whole, the principal multilevel quantization outperforms the other algorithms for a large set of test images. On the average, the quantization error produced by this algorithm is six times less than that of the median-cut algorithm and four times less than that of the variance-minimization algorithm.

IV. THE DYNAMIC ADAPTIVE COLOR QUANTIZATION ALGORITHM

We now describe our algorithm for dynamic adaptive color quantization of image sequences. The input to colormap design is the image histogram. To simplify computations and memory requirements, during histogram construction, color resolution of images is reduced from 8 to 5 bits per color component. Since the number of different colors in a typical image is significantly smaller than the number of pixels, quantizer mapping actually maps the image histogram buckets to color representatives. Postprocessing is required to map each pixel of the image to its quantized value. The preprocessing (histogram construction) and postprocessing are simple passes through image data, independent of the actual color quantization algorithm.

A. Colormap Design

We propose a heuristic method to find color representatives for the image. By iterative division of RGB space to cubes, a partial n -partition P of RGB is found. This partition induces an n -partition Q on the input image I with a sufficiently small value of $QRMSE(Q)$. The clusters of Q contain most pixels of I . The centroids of these clusters form the desired representatives. We call the divisive technique described below *oct-cut division* and the obtained RGB partition P and image partition Q *oct-cut partitions*.

At the beginning P contains only one element—the entire

RGB cube. At each iteration the cube in P containing the largest number of image pixels is chosen for division. This cube is divided to eight equal subcubes, by three orthogonal planes parallel to the cube faces and passing through their centers. Only those new subcubes which contain a sufficiently large number of image pixels are then included in P (this threshold is a heuristic parameter). The division is stopped after n cubes are obtained. The use of the bound on a number of image pixels that a cube must contain to be included in P reduces the value of $QRMSE(Q)$. Indeed, clusters of Q containing a very small number of pixels do not contribute much to the value of $QRMSE(Q)$, so do not influence the choice of the representative colors. Conversely, clusters with large population, hence with large contribution to the value of $QRMSE(Q)$, get more representatives in the colormap.

The oct-cut technique is located between the two standard families of algorithms—fixed and adaptive color quantization. It includes features of each of these techniques. Indeed, the choice of the cubes for division is based on the color distribution of the image in color space. However, division of each individual cube is independent of the color statistics inside this cube. The first property guarantees that the induced n -partition Q of the image has relatively small value of $QRMSE(Q)$. The second property provides two important characteristics:

- 1) Reduction of the computational cost of the partition process. The time complexity of the oct-cut technique is $O((n+m)\log n)$, where n is the number of cells in P and m the number of buckets in the image histogram.
- 2) Robustness of the oct-cut partition to small changes in the images. Oct-cut partitions of RGB obtained for similar images consist of many identical cubes.

The oct-cut technique is similar to the octree quantization method, mentioned in Section III.C.4. However, the oct-cut technique is a top-down procedure, in the sense that new cubes are generated, constantly expanding the tree. This contrasts with the octree method, which is bottom-down, in the sense that the tree is constantly being shrunk.

B. Colormap Filling

Let I_1, I_2 be two arbitrary consecutive images in the sequence and $Q_1 = \{b_1, b_2, \dots, b_n\}$, $Q_2 = \{c_1, c_2, \dots, c_n\}$ be their n -partitions induced by the corresponding RGB partitions $P_1 = \{t_1, t_2, \dots, t_n\}$, $P_2 = \{u_1, u_2, \dots, u_n\}$. Let $C_1 = \{q_1, q_2, \dots, q_k\}$, $C_2 = \{r_1, r_2, \dots, r_k\}$ be colmaps of I_1, I_2 . In general, $k \geq n$.

The intensity of the screen flicker between the display I_1 and I_2 is proportional to the value of $D_{I_1}(C_1, C_2)$ (see Section II.C). Since the value of $D_{I_1}(C_1, C_2)$ depends strictly on the order of the colors in the colormap C_2 , then, to reduce the screen flicker, the representative colors of I_2 must be assigned to the entries of C_2 in an order minimizing this.

The method we propose for screen flicker reduction is based on the following assumption about color statistics of consecutive images:

*Consecutive images in a video sequence
have very similar color statistics.*

It is possible to significantly reduce screen flicker by reserving in each colormap only a small number of entries for color representatives of the previous image. We call this set of colors *screen flicker compensators (SFC)*. The exact number of the reserved entries for the *SFC* colors depends on the dynamics of changing colors between consecutive images and can be chosen individually for each sequence (or changed on the fly).

When assigning colors to the colormap C_2 the following approach would provide the best results: First, color representatives of I_2 are assigned to the entries of C_2 in order to minimize the value of $D_{I_1}(C_1, C_2)$. Second, colors from C_1 , which correspond to the entries not assigned to C_2 ($k - n$ entries), form the set of *SFC* colors in C_2 . However, to find the global minimum of $D_{I_1}(C_1, C_2)$, we must check all of the $\frac{k!}{(k-n)!}$ different possibilities of placing the n representative colors in k entries of the colormap C_2 . For the most popular colormap sizes (more than 16 entries) this is prohibitively computationally expensive. So instead, we propose the following heuristic method, which significantly reduces the value of $D_{I_1}(C_1, C_2)$ in efficient time, while providing visually satisfactory results.

The assumption stated above implies (and this was confirmed in practice) that robust divisive techniques produce almost identical partitions of *RGB* for consecutive images. Thus, reduction of screen flicker may be obtained by retaining the entries of the representatives of the clusters induced by identical cells in the consecutive colormaps. This is the first step towards screen flicker reduction.

Assume we have found identical cells in P_1 and P_2 and placed the representatives of the clusters, induced by these cells, in the same entries of the colormaps C_1, C_2 (the first step). In this case the largest contributions to the value of $D_{I_1}(C_1, C_2)$ (and therefore to the screen flicker) are due to the representatives of the clusters induced by $p^1 p^2$ in Q_1 with the

largest weights in I_1 . In order to reduce this contribution, we add to C_2 entries from C_1 , which include the representatives of the heaviest clusters induced by $p^1 p^2$ in Q_1 . They are placed in the same entries as in the colormap C_1 . These colors constitute the set of *SFC* colors. Using this method, we completely eliminate the screen flicker in the pixels mapped to *SFC* colors. Fig. 2 shows the relation between partitions P_1 and P_2 in a simplified form.

C. Ordering Representatives in the Colormap

Assume the colormap C_1 for image I_1 is built. We want to fill the colormap C_2 with representatives of the clusters of Q_2 in an order minimizing the value of $D_{I_1}(C_1, C_2)$. Let $p = k - n$ be the number of entries reserved for *SFC* colors in each colormap. Filling colormap C_2 proceeds in three steps.

Step 1: Finding equal cubes

Let $Rp(s)$ denote the representative of the cluster s in the colormap. For each member $t_i, 1 \leq i \leq n$ of the set P_1 we search for an identical cube $u_j, 1 \leq j \leq n$ in the set P_2 . If such a cube is found, we assign $Rp(c_j) \leftarrow Rp(b_i)$, and the index of $Rp(b_i)$ in the colormap C_2 will be equal to its index in the colormap C_1 . This step guarantees that identical colors in successive colormaps occupy identical entries in those colormaps. To optimize the search for identical cubes we use hashing. The time complexity of Step 1 is $O(n)$.

Step 2: Finding SFC colors

In this step we choose p representatives in C_1 corresponding to the heaviest clusters of Q_1 induced by $p^1 p^2$ and place them at the same entries in the colormap C_2 . These chosen colors form the subset of *SFC* colors in the colormap C_2 . Since the *SFC* are located at the same entries in colormaps C_1 and C_2 , this prevents flicker in the pixels of I_1 quantized to these colors. The time complexity of Step 2 is $O(n)$.

Step 3: Completing the colormap

Let sfc_1 denote the set of entries of *SFC* colors in C_1 and A the set of entries of C_2 not yet assigned. We divide A into two subsets: $A_1 = sfc_1$ and $A_2 = A \setminus A_1$. The representatives placed in the entries of A_1 do not contribute to the screen flicker (since colors of C_1 at the entries of sfc_1 are not used in the color quantization of I_1). We then assign the representatives to the entries of A_2 (Step 3.1); and all remaining, not yet assigned, representatives (the most problematic colors) are placed in the entries of A_1 (Step 3.2).

Finding an optimal assignment to A_2 (minimizing $D_{I_1}(C_1, C_2)$ on these entries) may take $r!$ time, where $r = |A_2|$. In order to save computation time, we use a simple algorithm which does not, in general, provide the optimal assignment, but it is likely that the assignment it finds is close to optimal. The algorithm is outlined in Fig. 3.

The time complexity of Step 3.1 is $O(nr)$, the time complexity of Step 3.2 is $O(n)$.

After these three steps, all representatives of the clusters of Q_2 have been assigned and the colormap C_2 is complete.

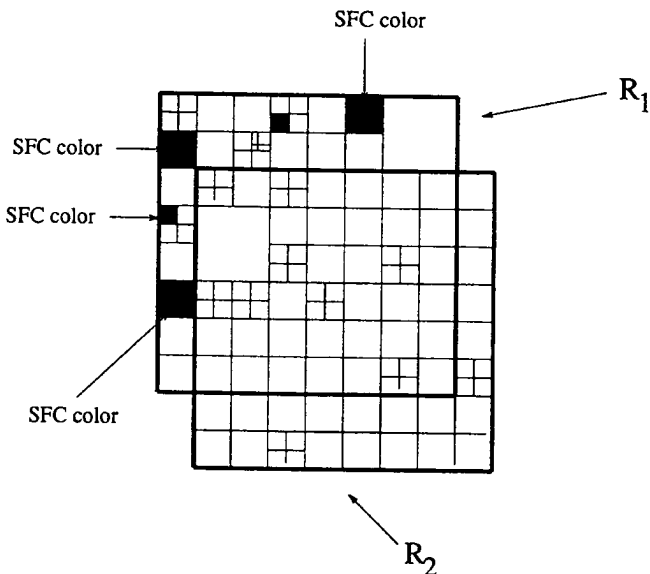


Fig. 2. 2D oct-cut quantization. Structure of consecutive partitions.

```

D = P2 \ P1
A1 = sfc1
A2 = A \ A1
While (A2 not empty) do
  Find an element j from A2 such that qj ∈ C1
    represents cluster with the maximal number of pixels;
  Find the cell ui in D such that the distance
    d(qj, Z(ci)) is minimal;
  Place Z(ci) at the j-th entry of colormap C2;
  A2 = A2 \ {j};
  D = D \ {ui};
end

```

Fig. 3. Assignment of the representatives of the clusters induced by $P_2 \setminus P_1$ to the entries of A_2 (Step 3.1).

D. Quantizer Mapping

For quantizer mapping, we use centroid mapping. We found in our simulations that the centroid mapping did not cause a significant increase in the *QRMSE* (relative to nearest representative mapping), yet significantly reduced computational time and provided quite reasonable image quality.

The oct-cut division algorithm (the first quantization task) allows the case of a small number of image pixels not belonging to any cell of the partition. Since, in typical images, adjacent pixel colors are highly correlated, a natural choice to quantize each of these pixels would be a representative color, assigned to a previously quantized neighboring pixel. In our algorithm, we test the left and above neighbors and select the one which is assigned a representative color closest to the color of the current pixel. The time complexity of centroid quantizer mapping is $O(1)$ per image histogram bucket.

V. EXPERIMENTAL RESULTS

A. Single Image Quantization

This section compares the quantization error and execution times achieved by the uniform, median-cut, octree and oct-cut quantization algorithms. The algorithms were tested on a number of 24-bit color public-domain images, including the 512×480 pixel 'Mandrill' image (Fig. 4a) and the 725×484 'Lighthouse' image (Fig. 6a). The quantization errors and execution times were measured on a Silicon Graphics VGX workstation, running C implementations of the algorithms. Uniform denotes uniform quantization, Median-cut NRM denotes median-cut quantization with nearest-representative quantizer mapping, Octree denotes octree quantization, Oct-cut NRM the oct-cut partitioning with nearest-representative quantizer mapping and Oct-cut CM the oct-cut partitioning with centroid quantizer mapping.

TABLE I
QRMSE VALUES OF DIFFERENT ALGORITHMS
ON THE "MANDRILL" TEST IMAGE

Colormap Size	Uniform	Median-cut NRM	Octree	Oct-cut NRM	Oct-cut CM
32	*	21.7	29.1	27.8	33.3
64	*	17.0	22.8	18.3	20.7
128	*	13.9	18.4	14.5	16.1
256	35.8	11.0	15.2	11.6	13.2

TABLE II
QRMSE VALUES OF DIFFERENT ALGORITHMS
ON THE "LIGHTHOUSE" TEST IMAGE

Colormap Size	Uniform	Median-cut NRM	Octree	Oct-cut NRM	Oct-cut CM
32	*	10.8	21.4	11.4	14.0
64	*	7.6	15.0	8.6	10.6
128	*	5.2	14.9	6.0	6.7
256	36.2	3.2	10.2	5.2	5.3

TABLE III
TIME (SECONDS) FOR COLORMAP DESIGN FOR MEDIAN-CUT
AND OCT-CUT ALGORITHMS ON THE "MANDRILL" IMAGE

Colormap Size	Median-cut	Oct-cut
32	0.05	0.02
64	0.08	0.03
128	0.09	0.03
256	0.11	0.04

TABLE IV
TIME (SECONDS) FOR COLORMAP DESIGN FOR MEDIAN-CUT
AND OCT-CUT ALGORITHMS ON THE "LIGHTHOUSE" IMAGE

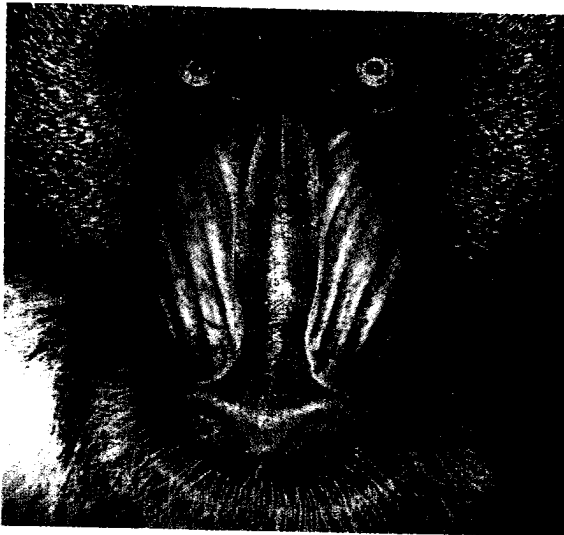
Colormap Size	Median-cut	Oct-cut
32	0.05	0.02
64	0.06	0.02
128	0.07	0.03
256	0.09	0.04

TABLE V
TIME (SECONDS) FOR QUANTIZER MAPPING FOR OCT-CUT NRM
AND OCT-CUT CM ALGORITHMS ON THE "MANDRILL" IMAGE

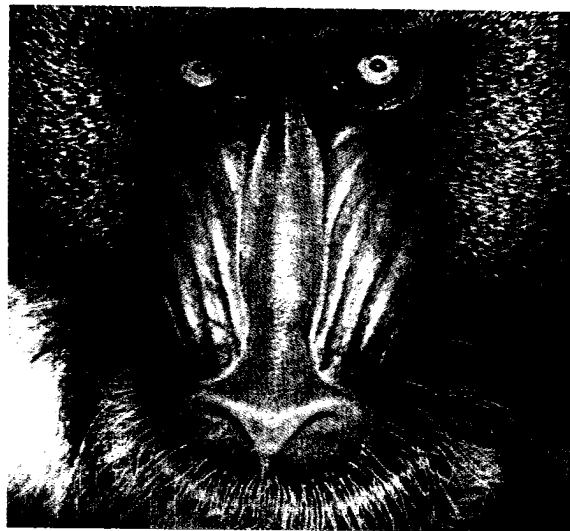
Colormap Size	Oct-cut NRM	Oct-cut CM
32	0.40	0.01
64	0.65	0.01
128	0.88	0.01
256	1.36	0.02

TABLE VI
TIME (SECONDS) FOR QUANTIZER MAPPING FOR OCT-CUT NRM
AND OCT-CUT CM ALGORITHMS ON THE "LIGHTHOUSE" IMAGE

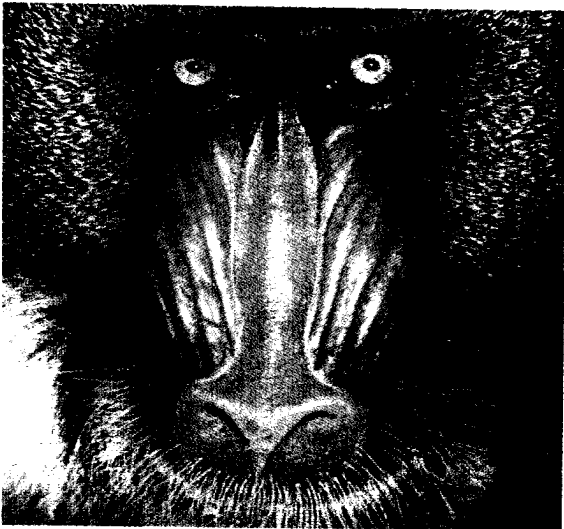
Colormap Size	Oct-cut NRM	Oct-cut CM
32	0.12	0.01
64	0.20	0.01
128	0.34	0.01
256	0.64	0.01



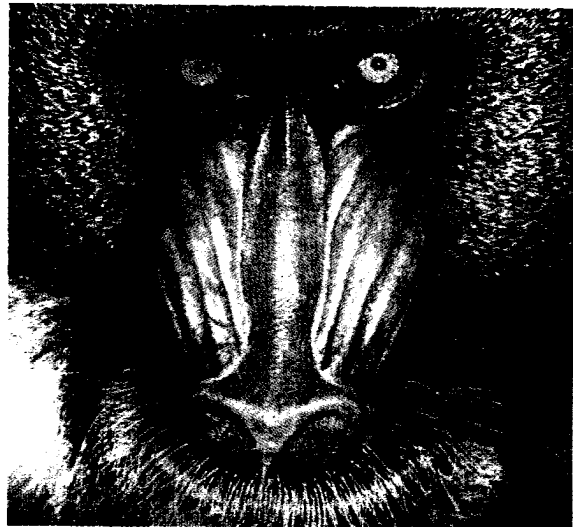
(a)



(a)



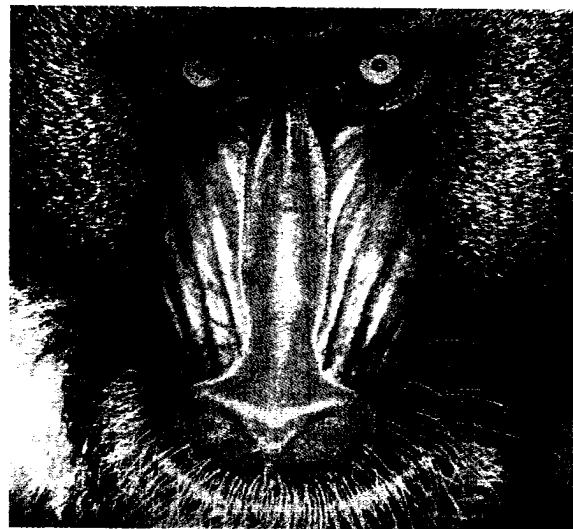
(b)



(b)



(c)



(c)

Fig. 4. (a) 24-bit RGB "Mandrill" image: 216,200 colors. (b) Uniform quantization to 256 colors: QRMSE: 35.8. (c) Median-cut NRM quantization to 256 colors: QRMSE: 11.0.

Fig. 5. "Mandrill" image: (a) Octree quantization to 256 colors: QRMSE: 15.2. (b) Oct-cut NRM quantization to 256 colors: QRMSE: 11.6. (c) Oct-cut CM quantization to 256 colors: QRMSE: 13.2.



(a)

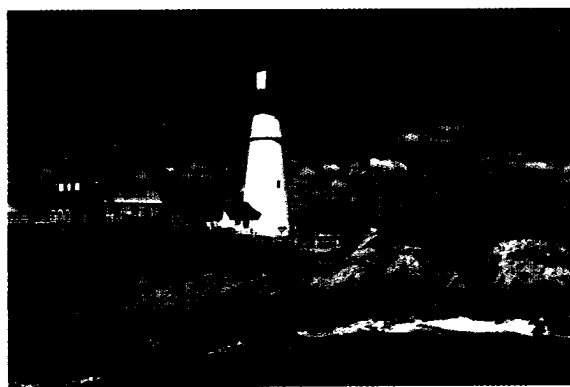


(b)

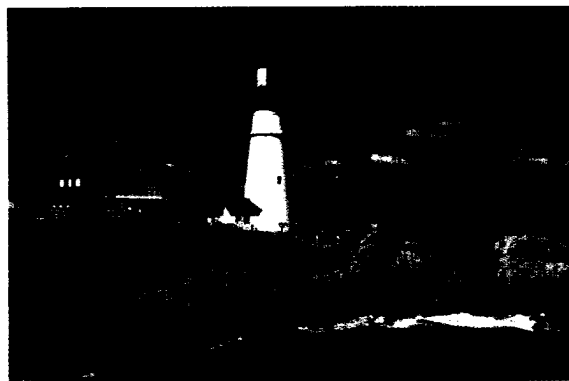


(c)

Fig. 6. (a) 24-bit RGB "Lighthouse" image: 79,881 colors. (b) Uniform quantization to 256 colors: $QRMSE$: 36.2. (c) Median-cut NRM quantization to 256 colors: $QRMSE$: 3.2.



(a)



(b)



(c)

Fig. 7. "Lighthouse" image: (a) Octree quantization to 256 colors: $QRMSE$: 10.2. (b) Oct-cut NRM quantization to 256 colors: $QRMSE$: 5.2. (c) Oct-cut CM quantization to 256 colors: $QRMSE$: 5.3.

Tables I, II, III, and IV show that for quantization to 256 colors (the standard colormap size on most graphic workstations), the oct-cut partition strategy is very effective. It reduces the time of colormap design by a factor of 2.8 for the "Mandrill" image and by a factor of 2.3 for the "Lighthouse" image relative to the median-cut algorithm, yet its $QRMSE$ values are only slightly larger than that of median-cut NRM, and considerably better than the values obtained by the uniform or octree methods.

Tables IV and VI compare time spent on quantizer mapping in oct-cut NRM vs. oct-cut CM. The results show that centroid mapping significantly reduces quantizer mapping time at the expense of a larger quantization error. However

the tradeoff favors time reduction, especially for large colormaps. For example, in the case of a 256 entry colormap, the increase in $QRMSE$ is 14% for the "Mandrill" image and 2% for the "Lighthouse" image, while the time of quantizer mapping is reduced by a factor of 68 for "Mandrill" and 64 for "Lighthouse". Hence, the use of centroid mapping as the quantizer mapping is extremely effective for real-time applications.

Figs. 4b, 4c, 5a, 5b, and 5c show the results of the five color quantization algorithms: uniform, median-cut NRM, octree, oct-cut NRM and oct-cut CM, applied to the "Mandrill" image. Figs. 6b, 6c, 7a, 7b and 7c show the results of the same algorithms applied to the "Lighthouse"

image. The uniform color quantization suffers from very severe degradation and contouring. The results of the four other algorithms are significantly better, of similar visual quality.

B. Dynamic Adaptive Color Quantization

The dynamic adaptive color quantization algorithm was tested on a number of animation sequences. The robustness and screen flicker reduction of the algorithm were measured on the "Table-tennis" color sequence. This sequence consists of 250 RGB images of resolution 360×240 pixels. The execution time of the basic functions of the dynamic adaptive color quantization algorithm was measured for live PAL video (576×768 pixels) digitized on a Silicon Graphics VGX workstation equipped with the *VideoLab* video I/O card.

In all the examples of this section, the size of a colormap was 256. The number of colormap entries reserved for SFC colors in dynamic adaptive color quantization of image sequences was 15, i.e., only 241 colormap entries were utilized for image quantization. The number 15 was obtained by trial and error, and seemed to be optimal for the images we tested.

C. Partition Robustness

The average number of identical cells in oct-cut partitions of consecutive images of the "Table-tennis" sequence was found to be 226. This confirms the robustness property of the partition strategy used in our algorithm. In contrast, the analogous number for the octree algorithm, the algorithm closest to ours in spirit, was only 65. This indicates that the octree algorithm would be less suitable for dynamic adaptive color quantization by our methods.

D. Screen Flicker Reduction

Screen flicker reduction is achieved during the "colormap filling" task of the algorithm. Fig. 8 shows the images of Fig. 1 as they appear during colormap switch using our algorithms. Without colormap filling, the value of $D_A(C_A, C_B)$ is 33. The use of colormap filling reduces $D_A(C_A, C_B)$ to 0, yielding an image much more visually pleasing than the first. Over the entire sequence (see Fig. 9), the average weighted distance is 33.5 without colormap filling, and 0.2 with it. In the latter case, the value of $D_j(C_j, C_{j+1})$ is zero for 83% of the "Table-tennis" images, meaning that screen flicker is eliminated entirely during display of these images. For most of the remaining 17% of the images the value of D is very small and screen flicker is unnoticeable to the human eye. The three maximal values of D greater than 4 correspond to a scene change.

E. Execution Times

The execution time of our algorithm was measured on a Silicon Graphics VGX workstation. The input RGB sequence was obtained by real-time digitization of a PAL analog video signal to a 576×768 pixel image sequence. To speed up histogram construction, a 4:1 spatial subsample of

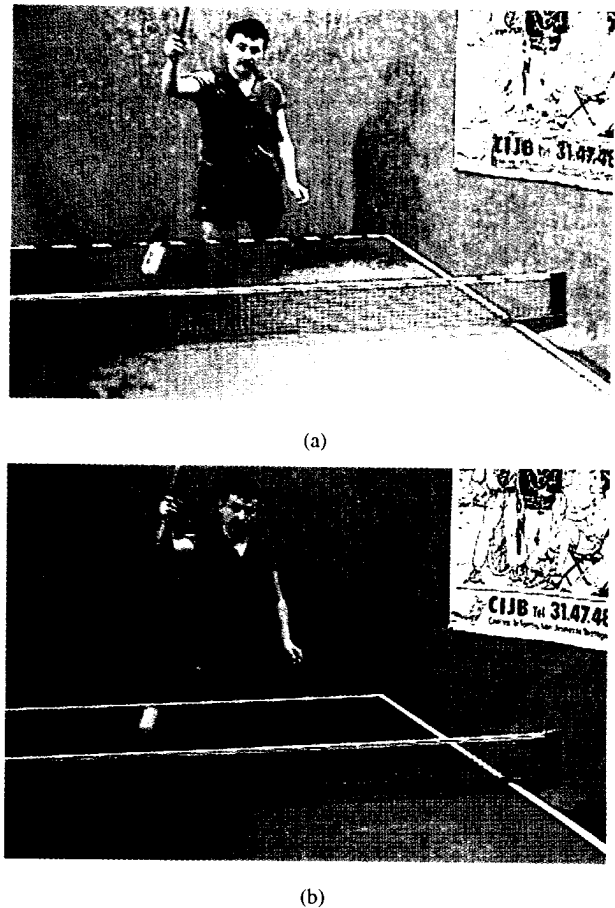


Fig. 8. An image from the "Table-tennis" sequence during colormap switch using oct-cut partitioning: (a) quantization without colormap filling, $D = 3.3$, (b) quantization with colormap filling, $D = 0$.

the image data was used. Table VII shows the results.

We used centroid mapping for quantizer mapping. Approximately 80% of the execution time was spent in the two simple passes through the image (preprocessing and postprocessing). The three quantization tasks (colormap design, colormap filling and CM quantizer mapping) require only 0.08 seconds per PAL image on the average. By implementing the passes of the image data in hardware, our algorithm may be applicable to real-time quantization of image sequences.

VI. CONCLUSION

A new algorithm for dynamic adaptive color quantization of image sequences has been proposed. The algorithm achieves excellent display quality due to individual quantization of each image to its own colormap and considerable reduction in screen-flicker. The fast oct-cut division technique, used for colormap design, produces a robust partition with small value of QRMSE. On the average, the oct-cut division technique outperforms (in run-time) the median-cut partition by a factor of 3, while producing approximately the same QRMSE. The robustness property of the oct-cut partition enables significant reduction of the distance between colormaps of consecutive images (quantifier of screen flicker) in linear time, using a small number of colormap

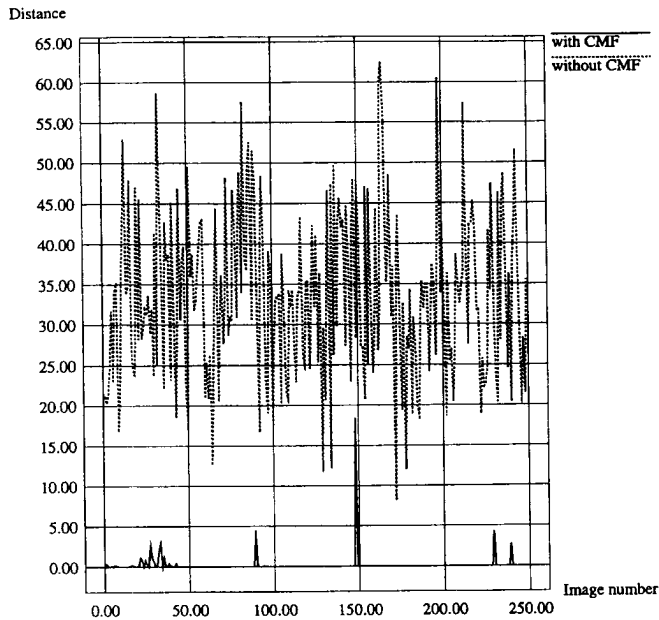


Fig. 9. Weighted distance between colormaps of consecutive images in the "Table-tennis" sequence.

TABLE VII
BREAKDOWN OF THE AVERAGE EXECUTION TIMES OF THE OCT-CUT CM ALGORITHM RUN ON A SEQUENCE OF PAL RESOLUTION IMAGES

Task	Execution time	% of total time
	per image (seconds)	
Preprocessing	0.11	27.5
Colormap design	0.06	15.0
Colormap filling	0.01	2.5
Quantizer mapping	0.01	2.5
Postprocessing	0.21	52.5
Total time	0.40	100

entries reserved for SFC colors. As a result, display of the quantized sequences is noise free. In our simulations we found that the centroid mapping is significantly faster than the optimal nearest-representative mapping, while the quality of the images remains comparable. The execution time of the three stages of the dynamic adaptive color quantization with CM quantizer mapping is 0.08 second per PAL resolution image. Special equipment, capable of real-time digitization of video images, combined with fast hardware implementation of the passes through image data during histogram construction and "image to histogram" mapping, would permit real-time implementation of the algorithm presented here.

REFERENCES

- [1] A. Gersho and R.M. Gray, "Vector quantization and signal compression," Kluwer Academic Publishers, 1992.
- [2] M. Orchard and C. Bouman, "Color quantization of images," *IEEE Trans. Signal Processing*, vol. 39, no.12, pp. 2,677-2,690, 1991.
- [3] P. Heckbert, "Color image quantization for frame buffer display" *Computer Graphics (SIGGRAPH)*, vol. 16, no. 3, pp. 297-307, 1982.

- [4] S.J. Wan, P. Prusinkiewicz, and S.K.M. Wong, "Variance-based colour image quantization for frame buffer display," *COLOUR Research and Application*, vol. 15, no. 1, pp. 52-58, 1990.
- [5] M. Gervautz and W. Purgathofer, "A simple method for color quantization: Octree quantization," A. Glassner, ed., *Graphics Gems*, pp. 287-293, Academic Press, New York, 1990.
- [6] P. Brucker, "On the complexity of clustering problems," *Optimization and Operations Research*, R. Henn, B. Korte, and W. Oetly, eds. Springer-Verlag, New York, pp. 45-54, 1977.
- [7] M.R. Garey, D.S. Johnson, and H.S. Witsenhausen, "The complexity of the generalized Lloyd-Max problem," *IEEE Trans. Information Theory*, vol. 28, pp. 255-256, 1982.
- [8] Y. Linde, A. Buzo, and R.M. Gray, "An algorithm for vector quantizer design," *IEEE Trans. Comm.*, vol. 28, pp. 84-89, 1980.
- [9] J.H. Friedman, J.L. Bentley, and R.A. Finkel, "An algorithm for finding best matches in logarithmic expected time," *ACM Trans. Math. Software*, vol. 3, no. 3, pp. 209-226, 1977.
- [10] N. Goldberg, "Colour image quantization for high resolution graphics display," *Image and Vision Computing*, vol. 9, no. 5, pp. 303-312, 1991.
- [11] R. Duda and P. Hart, *Pattern classification and scene analysis*. Wiley, New York, 1973.
- [12] G. Braudaway, "A procedure of optimum choice of a small number of colors from a large color palette for color imaging," *Proc. Electronic Imaging '87*, San Francisco, 1987.
- [13] S. Wan, S. Wong, and P. Prusinkiewicz, "An algorithm for multi-dimensional data clustering," *ACM Trans. Math. Software*, vol. 14, no. 2, pp. 153-162, 1988.
- [14] X. Wu, "Color quantization by dynamic programming and principal analysis," *ACM Trans. Graphics*, vol. 11, no. 4, pp. 348-372, 1992.
- [15] J.L. Furlani, L. McMillan, and L. Westover, "Adaptive color selection algorithm for motion sequences," *Proc. ACM Multimedia 94*, pp. 341-347, 1994.
- [16] E. Roytman and C. Gotsman, "Dynamic color quantization of video sequences," *Proc. Computer Graphics Int'l*, Melbourne, 1994.
- [17] J.M. Kasson and W. Plouffe, "An analysis of selected computer interchange color spaces," *ACM Trans. Graphics*, vol. 11, no. 4, pp.373-405, 1992.



Evgeny A. Roytman received the MS degree in mathematics from Moscow State Pedagogical Institute in 1986 and the MS degree in computer science from the Technion-Israel Institute of Technology in 1994.

From 1987 to 1989, Roytman was with the Mathematical Modeling Group at the Moscow Health Research Center, where he worked on the development of information systems and medical expert systems. Since 1993, Roytman has been with the Design Verification Group at the IBM Haifa

Research Laboratory where he is engaged in the development of automatic test generators for processor design verification.



Craig Gotsman earned his PhD in computer science from the Hebrew University of Jerusalem in 1990. Dr. Gotsman is now a senior lecturer in the Computer Science Department at the Technion-Israel Institute of Technology in Haifa, Israel and a consultant at the recently established Hewlett-Packard Israel Science Center in Haifa. His research interests are computer graphics and computational geometry, particularly the fields of rendering and animation.