

Color Reduction Using K-Means Clustering

Tomáš Mikolov*

Faculty of Information Technology
Brno University of Technology
Brno / Czech Republic

Abstract

Although nowadays displays are able to handle 24-bit graphics efficiently, it is still useful in many areas to reduce color space. The aim of this paper is to propose an easy to implement algorithm for color reduction using modified K-Means clustering. The comparison of results is made using popular ACDSee and Adobe Photoshop programs.

Keywords: Color reduction, K-Means clustering, dithering, mean square error

1 Introduction

It may not be obvious, but effective color reduction is needed in many graphical applications. One well known example may be the GIF format used widely on the Internet - this graphical format reduces the color space by defining a palette with size of 256 colors [5]. Another examples may be video codecs, computer games and various handheld devices like mobile phones.

Since the problem of finding optimal palette is computationally intensive (it is not possible to evaluate all possible combinations), many different approaches were taken to solve it, such as using neural nets, genetic algorithms, fuzzy logic, etc. On the other hand, for many applications it is much more appropriate to use simpler algorithms, like classic K-Means clustering.

The goal of this paper is to propose an easy to implement algorithm for color reduction with sufficient visual quality. The algorithm itself is described in chapters 3, 4 and the results and their comparison with output from standard programs (ACDSee 4.0 [7], Adobe Photoshop 6.0.1 [8]) are summarized in chapter 5.

2 Background

The problem of color reduction may be seen as a clustering problem. For example, for picture conversion to 16 colors, it is needed to find optimal positions of 16 clusters in RGB space that represent the input space so that the global error after picture conversion is minimized. Commonly used error function is mean squared error function (MSE, [6]), which can be simply

seen as an average of square of distances between input and output vectors.

Probably the simplest way to obtain these clusters is to cut off least significant bits in RGB values. For example, to reduce 24-bit picture to 8-bit, it is possible to cut off 5 least significant bits for red and green and 6 bits for blue. Although this algorithm is fast and simple, its results are insufficient for many applications.

Other common algorithms for color reduction are median cut algorithm [2], Wu color quantization algorithm [3] and neural network color quantization algorithm [4]. These algorithms differ in output quality, implementation difficulty and memory and time consumption. For example, solution using neural networks gives best results (in terms of MSE), but is computationally very expensive.

For many algorithms used in image processing and computer vision, implementation difficulty may be another important measure. It is clear that there is no 'best' algorithm for color reduction, since the solution always favours some characteristics (quality of output for neural networks, implementation difficulty for cutting off least significant bits).

One of the simplest algorithm for clustering is the classical K-Means algorithm [1]:

1. Choose K points (centroids) in space
2. Assign each input vector to the nearest centroid
3. Recalculate positions of all centroids, so that the new position of each centroid will be the average of all vectors that have been assigned to this cluster
4. Go to step 2 until the positions of centroids no longer move

What is not specified is the way how to choose the starting centroids in step 1. Usually, random selection from the input vectors is made.

Since this algorithm is very simple, it seems well suited to satisfy chosen demands.

3 Proposed Clustering Algorithm

The designed algorithm is a modification of the K-Means clustering. The difference is in the way how to obtain the initial positions of the clusters. Instead of random initialization, the clusters are built incrementally:

* xmikol04@stud.fit.vutbr.cz

1. Start with one cluster
2. Choose some number of vectors from input space at random; assign each of these vectors to the nearest cluster
3. Compute new centroids for all clusters
4. Find the cluster which was used the most; divide this cluster into two new clusters, move them in a random direction by a small amount
5. Go to step 2 until desired number of clusters is found

After this initialization, the clusters are adapted using classic K-Means algorithm.

4 Implementation

Implementation of the original K-means clustering is very straightforward. Suppose reduction to K colors: first, palette is established by choosing K random pixels from the input image. Second, K-means algorithm is used to modify this palette. Third, input image is converted to output image.

As it is computationally very expensive to perform step 2 in K-means algorithm mentioned in chapter 2, it is very useful to reduce the number of input vectors to some reasonable value. For testing purposes, all input images were 24-bit pictures with resolution 1024 x 1024 pixels. This would mean over million input vectors. It was observed that choosing 10, 000 randomly chosen input vectors from input image achieves good results with speedup factor of 100 (for this part of implementation). The second important thing is early stopping of the algorithm - it was observed that 40 iterations is enough. Although the centroids keep changing for a long time, no significant improvement is achieved after 40. iteration. This means that the computational complexity is constant - it may be useful to choose more input vectors for larger images, but since the value itself is a compromise between quality and speed, it is not necessary.

Conversion of input image to output image is very simple. For all input pixels, the nearest cluster is found by linear search over all clusters. The value of the output pixel is then the position of this cluster. This part of the implementation is very simple and ineffective, and large speedups can be gained by performing other than linear search. For example, simple hashing may be used to prevent recomputation of pixels at the same position in the input space. This results in significant speedups for images with large areas with exactly the same color. However, reported results are without any optimizations in this part of implementation (to prevent confusion of the reader). So the complexity is $S*K$, where S means the size of the input image in pixels and K means the desired number of colors in the output image.

Although the K-means algorithm itself works pretty well for color reduction problem, its modification as is mentioned in chapter 3 was investigated to see whether better than random initialization leads to better results.

The difference in implementation lies in the part before the K-means algorithm itself - instead of randomly choosing the starting palette from the input picture, the starting palette is incrementally built as is described in chapter 3. The number of vectors in step 2 was chosen to be 1 000 (using testing data). It is useful to discard from the splitting process new clusters (four last clusters in the current implementation). In step 4, new clusters are moved in opposite directions by a random vector with size 30. Note that these values were determined by using testing data set; it is possible that for some class of pictures, other values may work better. Computational complexity of this part of implementation is determined only by number of clusters.

Memory consumption of whole implementation is determined just by the size of input image and palette - $2*S + 3*K$, where S is the size of input image and K is the number of clusters.

Simple dithering method was implemented for better visual quality and possibility of comparison with ACDSee program that does not allow color reduction without dithering. Note that dithering always increases the MSE value, and various implementations behave differently.

Because this work is not primarily focused on dithering, only simple method based on transferring part of the error after pixel conversion to its right neighbour was used.

5 Results

Popular Adobe Photoshop (version 6.0.1) and ACDSee (v. 4.0) programs were used for comparison with output from K-means and modified K-means algorithms using MSE. Since ACDSee does not support color reduction without dithering, reported results include those obtained with simple dithering method mentioned in chapter 4. Since the algorithms use random number generator, the results vary with each conversion - to avoid this noise, reported results are computed as an average for 50 conversions for each picture.

Sample pictures obtained by reduction to 16 colors are included in appendix A. Results for reduction to 256 colors are reported, but the pictures themselves are not included, since the image distortion is almost invisible.

<i>Picture</i>	<i>K-means</i>	<i>modified K-means</i>	<i>m.K-means + dithering</i>
1	909	894	1467
2	382	372	641
3	355	349	534
4	511	508	891

Table 1: MSE on sample pictures using K-means, reduction to 16 colors

Modified version of K-means clustering indicate a modest improvement over basic version.

<i>Picture</i>	<i>Photoshop</i>	<i>Photoshop +dithering</i>	<i>ACDSee</i>
1	933	1477	2081
2	453	594	1731
3	411	531	755
4	502	688	1270

Table 2: MSE on sample pictures using popular programs, reduction to 16 colors

Tables 1 and 2 may be used for direct comparison of results between Adobe Photoshop and K-means clustering. Surprisingly, the simple implementation of K-means is significantly better at pictures 2 and 3. Results for pictures 1 and 4 are quite similar. Results obtained with dithering show quite bad performance of ACDSee.

<i>Picture</i>	<i>K-means</i>	<i>modified K-means</i>	<i>K-means +dithering</i>
1	100	103	224
2	57	54	116
3	58	60	114
4	59	62	122

Table 3: MSE after reduction to 256 colors

Results from table 3 show that there is no gain in using modified version of K-means for reduction to 256 colors.

<i>Picture</i>	<i>Photoshop</i>	<i>Photoshop +dithering</i>	<i>ACDSee</i>
1	132	172	292
2	75	94	146
3	85	105	159
4	92	118	193

Table 4: MSE after reduction to 256 colors using popular programs

Direct comparison of results in tables 3 and 4 confirms superiority of K-means clustering algorithm over results obtained with Adobe Photoshop and ACDSee even for reduction to 256 colors.

The only weakness seems to be the computational complexity. Both Adobe Photoshop and ACDSee maintain time for conversion of testing image (24 bit, resolution 1024 x 1024) to be far less than a second (approximately 0.1 second). For reduction to 16 colors, K-means finishes in 0.4 sec, while for reduction to 256 colors, it's about 4 seconds for original and 4.5 for modified method (@ AMD Sempron 2200+). This is caused by simple implementation of nearest cluster search (see chapter 4 for details). The overall speed of the implementation may be heavily increased by using better than linear search over all clusters.

Although MSE is an objective function, it is not the only measure we are interested in when comparing results

from different algorithms. In many cases, subjective appearance of the image may be more important.

First picture (see appendix A) consist of four pictures (don't get confused!). The purpose of this is to test algorithms on a picture with wide variability of colors. Best results were obtained with modified K-means (for 16 colors) and original K-means (for 256 colors). Adobe Photoshop maintained good results, while ACDSee performed poorly - there can be seen a rather big distortion in color of the sky.

The second picture is a cartoon picture. During testing, best improvement by using modified version of K-means clustering against original version was achieved on this type of pictures (up to 40% for pictures which had already reduced palette size). Again, K-means is the best, Photoshop maintains good results and ACDSee is the worst one.

The third and fourth pictures show again rather poor performance of ACDSee, while Photoshop maintains comparable results.

6 Conclusion and future work

Although very simple, K-means algorithm seems to be very useful for color reduction problem. Achieved results are better than those obtained by using popular Adobe Photoshop and ACDSee programs.

Modification of the initial phase of the algorithm was investigated with rather modest improvement for reduction to 16 colors and no improvement for reduction to 256 colors. However, for some sort of pictures (those containing already somewhat reduced palette size), much better improvements were achieved. This may be investigated in the future.

Future work should be focused on improving the overall speed by using better search techniques for finding the nearest cluster.

The algorithm itself may be very well used in a preprocessing phase before more advanced algorithms from image processing and computer vision.

References

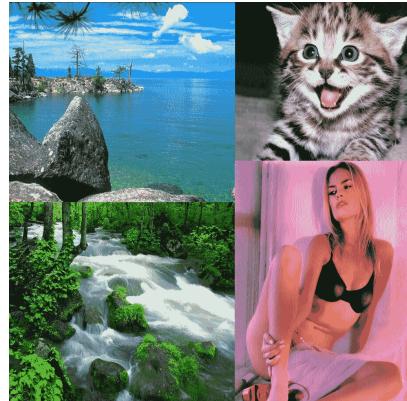
- [1] J. B. MacQueen (1967): "Some Methods for classification and Analysis of Multivariate Observations, Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability", Berkeley, University of California Press, 1:281-297
- [2] P. Heckbert (1982): "Color Image Quantization for Frame Buffer Display", Computer Graphics, vol. 16, No. 3, pps. 297-307.
- [3] X. Wu (1991): "Efficient Statistical Computations For Optimal Color Quantization", in Graphics Gems, vol. II, edited, by James Arvo, Academic Press, Inc., Cambridge, MA, pp. 126-133.

- [4] A. Dekker (1994): “Kohonen neural networks for optimal colour quantization”, Network Computation in Neural Systems, Vol. 5, pps. 351-367.
- [5] <http://en.wikipedia.org/wiki/GIF>
- [6] http://en.wikipedia.org/wiki/Mean_squared_error
- [7] ACDSee portal. <http://www.acdsee.com/>
- [8] <http://www.adobe.com/products/photoshop/>

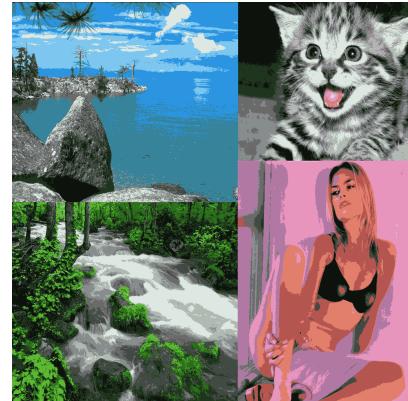
Appendix A - sample pictures for reduction to 16 colors:



a) Original picture



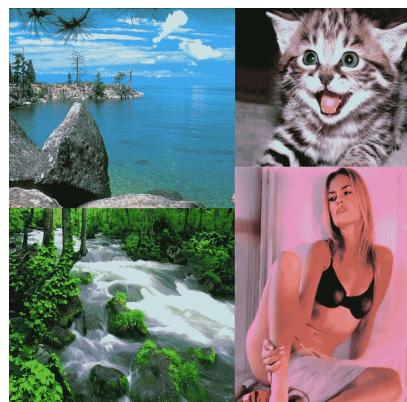
b) K-means with dithering



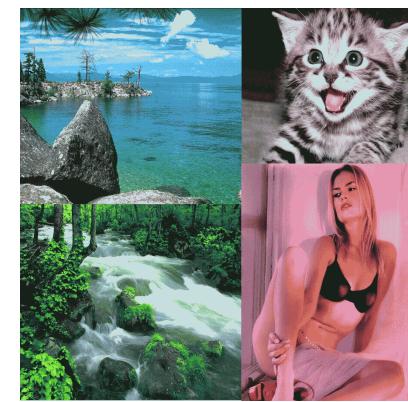
c) K-means



d) Photoshop no dithering



e) Photoshop with dithering



f) ACDSee (dithering)

Figure 1 - sample picture consisting of four pictures



a) Original picture



b) K-means with dithering



c) K-means



d) Photoshop no dithering



e) Photoshop with dithering

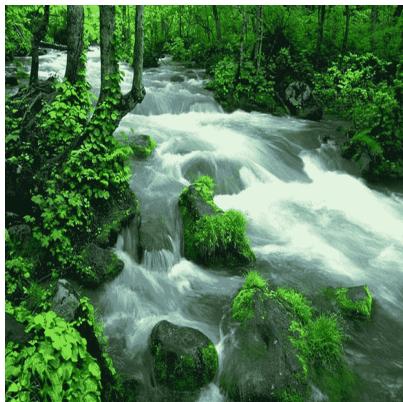


f) ACDSee (dithering)

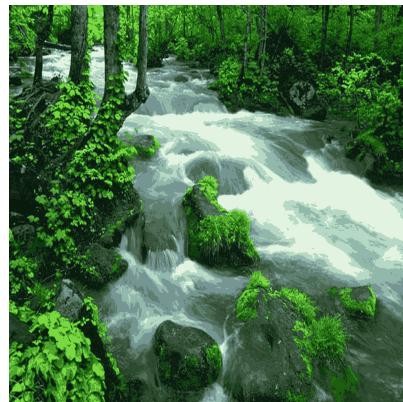
Figure 2 - sample cartoon picture



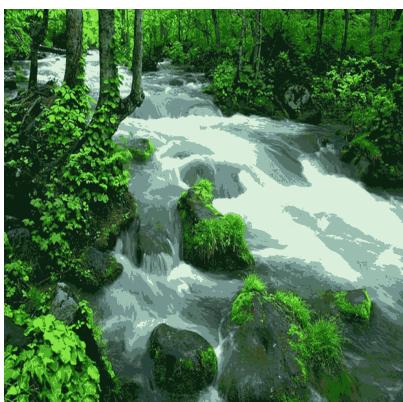
a) Original picture



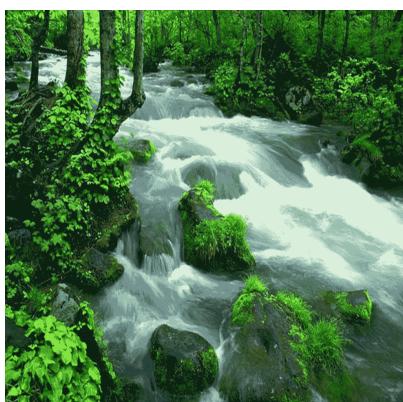
b) K-means with dithering



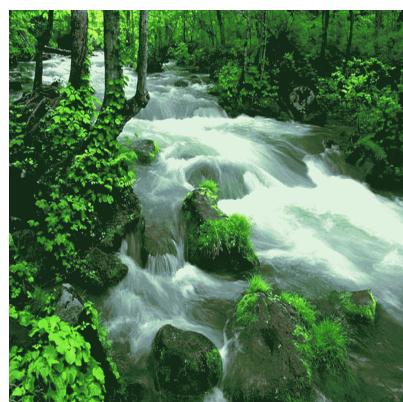
c) K-means



d) Photoshop no dithering

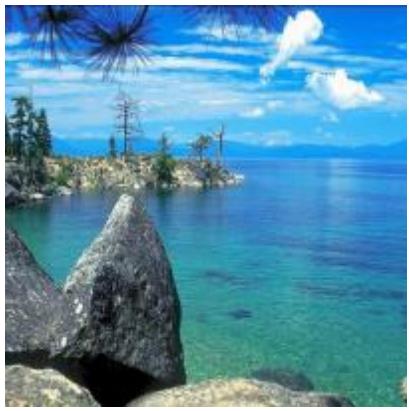


e) Photoshop with dithering

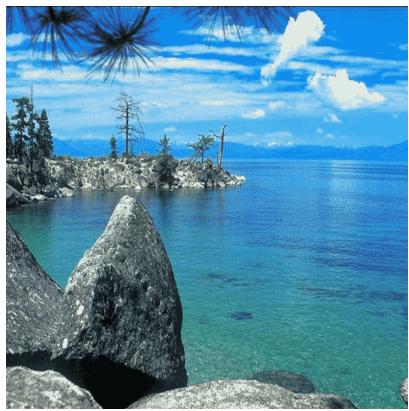


f) ACDSee (dithering)

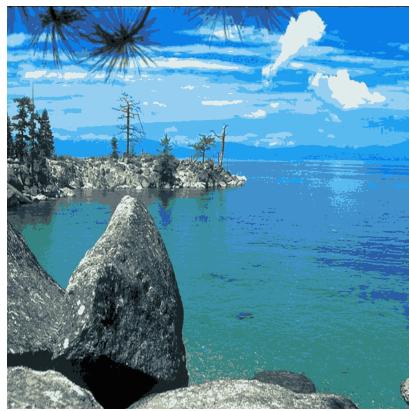
Figure 3



a) Original picture



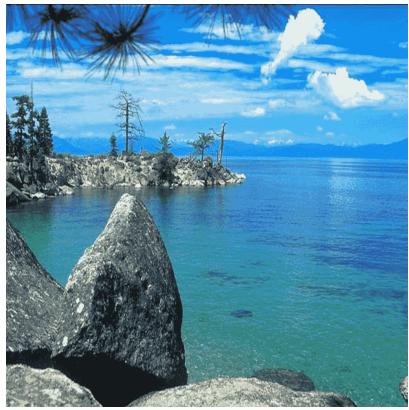
b) K-means with dithering



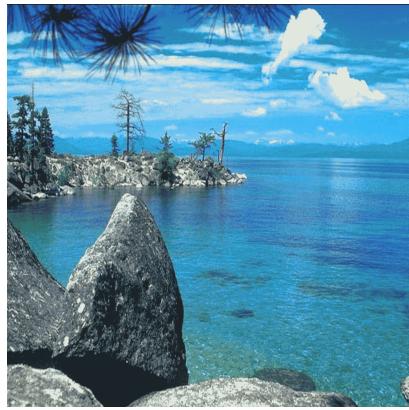
c) K-means



d) Photoshop no dithering



e) Photoshop with dithering



f) ACDSee (dithering)

Figure 4