Vector Quantization Based on Block Orientation [*]

Chia-Hao Yang
Department of Applied Mathematics

Chang-Biau Yang
Shyue-Horng Shiau
Department of Computer Science and Engineering,
National Sun Yat-sen University, Kaohsiung, Taiwan
Email:cbyang@cse.nsysu.edu.tw

Abstract

Vector quantization (VQ) is an effective method of data compression. Compared with other image compression techniques, VQ has a high compression ratio and a simple and low computation complexity. So VQ has a good performance in image compression. However, VQ needs much time in the encoding phase. For keeping high quality of image, VQ needs full searching to find the closest codeword in the encoding phase. In this paper, we propose a fast algorithm to reduce the encoding time in VQ. This algorithm is based on the block orientation. By the experiments, our algorithm need less time in the encoding phase, and the quality is comparable with the VQ with the full searching encoding in the same compression ratio.

1 Introduction

The concept of data compression has been pointed out for a long time and has been studied in many areas such as images, texts, speech, data and so on. Data compression can be used to reduce the size of data in the storage. Image compression is a principal part of data compression because it is developed extensively in many applications such as video conference, image database, medical image and so on. In the image compression, we can remove the insignificant information from the images. The compressed images are tested by the human visual system, so the quality of the images depends on the perceptual entries such as the contours of regions and edges. Thus, we can get a high compression ratio by reducing the insignificant and redundant information from the image. Many image compression methods have been proposed, such as differential pulse code modulation (DPCM) [18] and block truncation coding (BTC) [2, 3, 13, 19], and the vector quantization (VQ) [4, 8–12, 14–17]. Although DPCM and BTC are methods of high-quality compression, the compression ratio is not good enough. So VQ, which has high compression ratio, is more popular than DPCM and BTC.

In the VQ technique, the codebook has a great effect upon the image quality. Therefore, a good algorithm for generating codebook is very important for VQ. There are several well-known codebook generation algorithms, such as *Linde-Buzo-Gray algorithm* (LBG) [10], the *maximum descent* [1, 11] (MD) algorithm and the longest distance distance first (LDF) algorithm [5].

In this paper, we propose a new method used in the codebook generation phase and the encoding phase. This method is based on the block orientation. Including this concept, we propose our algorithm for vector quantization. Our algorithm consists of three phases, generating codebook, encoding and decoding, which is the same as the conventional VQ. By our experiments, the performance of our algorithm is more efficient than the conventional VQ with the same compression ratio, and the quality of the reconstructed image is also better than the conventional VQ.

In Section 2, we will introduce the vector quantization and present the codebook generation algorithm that we used in this paper. In Section 3, we will propose our image coding algorithm with VQ. In Section 4, we will show the performance of our algorithm and compare it with the conventional VQ. Finally, the conclusion will be given in Section 5.

[*]This research work was partially supported by the National Science Council of the Republic of China under contract NSC-88-2213-E-110-012.

2 Definitions and Notations

In the conventional VQ, an image is divided into a set of blocks v_j with size $\sqrt{k} \times \sqrt{k}$. We call these small blocks *vectors*. Suppose the codebook size is N. Then we first have to generate a codebook $B = \{b_i \mid i = 1, 2, ..., N\}$ of k-dimensional codewords from the set of training vectors. Then, in the encoding phase, we compare each original blocks v_j taken from the original image with all codewords in B. Each original vector v_j will have a closest codeword b_i based on the minimum square error (SE) and we can use the index of b_i to represent v_j. That is, v_j can be represented by an index i, $1 \leq i \leq N$. When we want to use or manipulate the image, we have to restore the image (the decoding phase). In the decoding phase, v_j is simply restored by b_i. The definition of square error distortion is $D(v_x, v_y) = \sum_{j=1}^{k} |v_{x_j} - v_{y_j}|^2$, where v_x and v_y are two k-dimensional vectors.

In general, it is believed that we need the longer computation time to generate the superior codebooks with the LBG [10] algorithm and the fast algorithm of codebook generation usually suffers from slight degradation in overall distortion. However, the maximum descent (MD) [1,11] algorithm, proposed by Chok-Ki Chan and Chi-Kit Ma, is one of the efficient methods for generating vector quantization codebooks. Compared with the LBG algorithm, the MD algorithm has lower overall distortion errors and needs less computation time.

Suppose that we shall design a codebook with size N and we are given a training set $V = \{v_m \mid m = 1, 2, ..., M\}$, consisting of M k-dimensional vectors (blocks with size $\sqrt{k} \times \sqrt{k}$) where $M \gg N$. At the first step, the MD algorithm treats the training vector set V as a global cluster. Then the method partitions this global cluster into two new clusters by a certain partition method. One of these two clusters is selected to be partitioned into two new clusters according to the *maximum distortion reduction criterion*. Now, the number of clusters is three. We continue this process to select one of the clusters to split into two new non-empty clusters until the number of clusters is equal to N. Then, the centroid of each cluster represents one codeword of the codebook. Thus, the codebook with N codewords is built.

In the following, let us show more details of the MD algorithm. Suppose that cluster C_i is partitioned into two non-empty clusters C_{ia} and C_{ib}. Let $R(C_i)$ denote the total distortion error in cluster C_i. That is, $R(C_i) = \sum_{v_i \in C_i} D(v_i, \bar{v}_i)$, where v_i is the vector in C_i and $\bar{v}_i$ is the centroid of C_i. Then the reduction of distortion error $\triangle R_i$ due to partitioning cluster C_i into C_{ia} and C_{ib} is $R(C_i) - [R(C_{ia}) + R(C_{ib})]$, i.e., $\triangle R_i = R(C_i) - [R(C_{ia}) + R(C_{ib})]$.

Now, consider the general case of S clusters. The MD algorithm wants to find $S+1$ clusters by partitioning one of the current clusters into two new non-empty clusters and keeping the other clusters unchanged such that we can get a maximum distortion reduction. At this time, the distortion reduction functions of all clusters have been computed. Cluster C_j is selected to be split into two new clusters C_{ja} and C_{jb} if $\triangle D_j \geq \triangle D_i$ for $1 \leq i, j \leq S$. Then we have $S+1$ clusters. When we continue splitting these clusters, we need only to calculate $\triangle D_{ja}$ and $\triangle D_{jb}$ of the newly formed clusters C_{ja} and C_{jb}. Since $\triangle D_i$'s of other clusters have been computed in the previous steps. So, to form N clusters, $N-1$ splitting operations and $2N-3$ partition searches (to find the value of $\triangle D$) are required.

The partition is deeply dependent on the distortion reduction, and the quality of the codebook generated by the MD algorithm is dependent on the splitting technique. Thus, the main part of the MD algorithm is how to partition one cluster into two new clusters. There are several methods to perform the partition, such as searching the optimal partitioning hyperplane, the 2-level LBG algorithm and the longest distance partition (LDP) method [5].

3 Vector Quantization Based on Block Orientation

In this section, we will propose a new vector quantization method, based on the block orientation. Our idea is extracted from some features of fractal image coding [6, 7]. Before describing our algorithm, we shall first define the block orientation rules. And in this section, we treat the vector discussed in the previous section as a block form. In other words, a 16-dimensional vector is viewed as a 4×4 block.

Suppose that Figure 1(a) and (b) are two distinct blocks extracted from the image. When we compare them without any rotation, we know that they are very different. However, if a rotation is included, it is clear that Figure 1(b) is the same as Figure 1(a) by rotating Figure 1(b) with 180°. Thus, when we calculate the similarity or distance of two blocks, we will try to rotate one of the two blocks with some angles.

The eight orientations used here are as follows: 0° rotation (origin), 90° rotation, 180° rotation, 270° rotation, and reflections of the *horizontal midline*, *vertical midline*, 45° line, and 135° line. When the eight orientations are involved, any two vectors X and

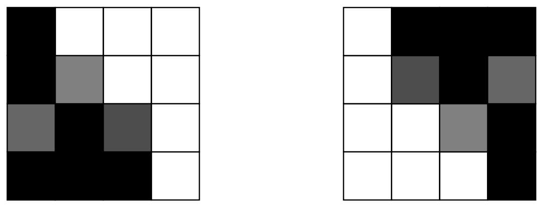

Figure 1: Block rotation.

Y have eight distances, defined as follows:

$$D_j(X,Y) = \sum_{i=1}^{k}(O_j(x_i) - y_i)^2, 1 \leq j \leq 8,$$

where O_j is the jth orientation function. Thus, the distance of these two vectors is redefined as $Dis(X,Y) = min_{j=1}^{8}\ D_j(X,Y)$.

Our algorithm is one kind of VQ based on the block orientation technique. Our method still has three phases: codebook generation, encoding and decoding. In codebook generation phase, we still apply the MD algorithm with the 2-level LBG. But our partition method involves the block orientation concept.

Originally, when the distance of two vectors (blocks) X and Y is calculated, $Dis(X,Y)$ has to be performed. Thus, eight orientations are involved. However, the calculation of eight orientations spends much time. From our experiments, which will be given in the next section, we can modify our concept to make it more efficient. In the codebook generation phase, we find that the blocks need not to be rotated in all of eight orientations when they are compared with the centroid. When a cluster is partitioned (This is called a partition step.), several iterations are needed. For each partition step, the partition iteration terminates until a predefined threshold is reached. We find that most blocks do not switch from the current cluster to the other cluster after the first iteration in each partition step. In other words, the orientation of most blocks is almost fixed. Therefore, in each partition step, eight orientations are calculated only for the first iteration, and only one orientation is used after the second iteration. Our partition method is as follows:

Algorithm Orientation-Partition

Input: One cluster C_i.

Output: Two new clusters C_{ia} and C_{ib} split from C_i.

Step 1: Find the centroid of cluster C_i, say $\bar{v}_i$. Let $\bar{v}_{ia} = \bar{v}_i$ and $\bar{v}_{ib} = \bar{v}_i + I$, where I is the vector with all elements being 1.

Step 2: For each vector v in C_i, calculate $Dis(v, \bar{v}_{ia})$ and $Dis(v, \bar{v}_{ib})$.

Step 3: If $Dis(v, \bar{v}_{ia}) < Dis(v, \bar{v}_{ib})$, then v belongs to C_{ia}, otherwise v belongs to C_{ib}.

Step 4: Calculate the new centroids $\bar{v}_{ia}$ and $\bar{v}_{ib}$ of clusters C_{ia} and C_{ib}, respectively.

Step 5: For each vector v in C_i, calculate $D_{j_v}(v, \bar{v}_{ia})$ and $D_{j_v}(v, \bar{v}_{ib})$, where j_v is the orientation index of v in Step 3.

Step 6: If $D_{j_v}(v, \bar{v}_{ia}) < D_{j_v}(v, \bar{v}_{ib})$, then v belongs to C_{ia}, otherwise v belongs to C_{ib}.

Step 7: Repeat Steps 4-6 until the distortion error $\triangle R_i < h$, where h is a predefined threshold.

In the codebook generation algorithm, we apply our Orientation-Partition technique to split one cluster into two clusters.

Algorithm G

Input: The training set $V = \{v_i \mid i = 1, 2, ..., M\}$ and the codebook size N.

Output: The codebook $B = \{b_i \mid i = 1, 2, ..., N\}$, the codeword index and the orientation index of each vector v_i, $1 \leq i \leq M$.

Step 1: Let V be a global cluster C. Apply Algorithm Orientation-Partition to partition C into two clusters.

Step 2: Select the maximum $\triangle R$ among all clusters. And partition the maximum one into two new non-empty clusters by applying Algorithm Orientation-Partition.

Step 3: Repeat Step 2 until the number of clusters is N.

Step 4: Calculate the centroid c_i of each cluster C_i, and assign $b_i = c_i$, where $1 \leq i \leq N$.

In the codebook generation phase, each vector must compare to the centroid of the cluster with our orientation rule. And when the vector compares to the centroid of the cluster, we need record the orientation of the vector and the index of the cluster to which the vector belongs.

In the encoding phase, we need not compare each vector with each codeword. When we generate the codebook, we have already determined that each vector is represented by which codeword. However, we need more memory to store encoded information, since we have to record the orientation of each vector. Thus, the following information has to be stored: the codebook, the index of each vector, and the orientation index of each vector which we use to encode.

In the decoding phase, we use the index and orientation of each vector and the codebook to decode the image. We find the correct codeword from the codebook by the index of the decoding vector and rotate the codeword to the correct orientation by the orientation index of the decoding vector. After each vector is decoded, the image is restored.

From our experiments and performance analysis, which will be given in the next section, we further find that the blocks need not to be rotated in all of eight orientations when they are compared with the centroid in each partition step. The orientation of each block may be fixed after some partition steps. We can check the orientation of each block whether or not it is the same as that in the previous partition step. If the orientation of one block is the same as that in the previous partition step, then we do not rotate it any longer after this partition step. We call this version Algorithm A, which is described as follows.

Algorithm A

Input: The training set $V = \{v_i \mid i = 1, 2, ..., M\}$ and the codebook size N.

Output: The codebook $B = \{b_i \mid i = 1, 2, ..., N\}$, the codeword index and the orientation index of each vector v_i, $1 \leq i \leq M$.

Step 1: Let V be a global cluster C. Apply Algorithm Orientation-Partition to partition C into two clusters.

Step 2: Select the maximum $\triangle R$ among all clusters. And partition the maximum one into two new non-empty clusters by applying Algorithm Orientation-Partition. When Algorithm Orientation-Partition is applied, for each vector v in the selected cluster, if its orientation is the same as that in the previous partition step, then replace Dis by D_{j_v}.

Step 3: Repeat Step 2 until the number of clusters is N.

Step 4: Calculate the centroid c_i of each cluster C_i, and assign $b_i = c_i$, where $1 \leq i \leq N$.

From some experiments, shown in the next section, we further find that the orientation of some blocks may be fixed in the first two partition steps. However, this may get poor image quality. And if the orientation is fixed after three partition steps, all images have good quality. Thus, we further modify our algorithm to get Algorithm B by adding one more condition: each block has to rotate in all of eight orientations at the first three partition steps when it is compared with the centroid. Algorithm B is as follows.

Algorithm B

Input: The training set $V = \{v_i \mid i = 1, 2, ..., M\}$ and the codebook size N.

Output: The codebook $B = \{b_i \mid i = 1, 2, ..., N\}$, the codeword index and the orientation index of each vector v_i, $1 \leq i \leq M$.

Step 1: Let V be a global cluster C. Apply Algorithm Orientation-Partition to partition C into two clusters.

Step 2: Select the maximum of $\triangle R$ among all clusters. And partition the maximum one into two new non-empty clusters by applying Algorithm Orientation-Partition. When Algorithm Orientation-Partition is applied, for each vector in the selected cluster, if its orientation is the same as that in the previous partition step and the partition depth is greater than 3, then replace Dis by D_{j_v}.

Step 3: Repeat Step 2 until the number of clusters is N.

Step 4: Calculate the centroid c_i of each cluster C_i, and assign $b_i = c_i$, where $1 \leq i \leq N$.

4 Experiments and Performance Analysis

In this section, we will show our experiments and analyze the performance of our algorithms. Our testing images are Lena, Baboon, F16, and Pepper of size 512×512 with 256 gray levels. The testing environment is performed on an IBM compatible PC with AMD-K6 CPU (Pentium compatible) and 96 MB RAM, and the compiler of the simulation programs is Borland C++ Builder.

The distortion of the encoded image is measured by the peak signal-to-noise (PSNR), which is defined as PSNR= $10\log_{10}[\frac{255^2}{\frac{1}{L\times L}\sum_{i=1}^{L}\sum_{j=1}^{L}(x_{ij}-\hat{x}_{ij})^2}]$, where $L \times L$ = size of image, x_{ij} = pixel value of the original image at coordinate (i, j), and $\hat{x}_{ij}$ = pixel value of the reconstructed image at coordinate (i, j) [1].

Table 1 shows the performance of our partition algorithm. In Table 1(a), we find if the partition depth (number of iterations in each partition step) is greater than three, the encoded image quality has only little improvement. And when the partition depth increases, the execution time also increases, as shown in Table 1(b). It is the reason why we do some modification of our algorithm to get Algorithm B.

Table 2 shows the performance of our modified algorithms, where the image size is 512×512 and the codebook size is 256. In Table 2(a), when we generate the codebook, if the vector's orientation is the same as that in the previous partition step, then we will not rotate it any more. According to our experiments, the vector's orientation may have large change only in the first three partition steps. In Table 2(b), in addition to Table 2(a)'s condition, we also stipulate that each vector has to rotate at least three times. The bit-rate per pixel (bpp) is 0.8125.

From Table 3 and Table 2, we can compare the performance of our algorithm and the conventional VQ under the same bpp. Compare our algorithm with the conventional VQ under 0.8125 bpp, in the testing images of size 512×512, the quality of reconstructed images of our algorithm is better than that of the conventional VQ, and the total execution time of our algorithm is about 1/3 of the conventional VQ.

5 Conclusion

In this paper, we propose a fast algorithm for image vector quantization based on block orientation. In the encoding phase, the conventional VQ spends much time to find the best codeword for each block to preserve the quality of reconstructed image. Our algorithm need not do anything in the encoding phase by increasing a little time in the codebook generation phase with the block orientation concept. With the same codebook size, the quality of image reconstructed by our algorithm is much better. However, our algorithm needs more memory to record the additional information, the orientation index of the block. Thus, when we compare the performance of our algorithm and the conventional VQ, different codebook sizes are used. By the experiment results under the same bpp, in the testing images of size 512×512, the quality of reconstructed images of our algorithm is better than that of the conventional VQ, and the required time is reduced to about 1/3. Thus, the block orientation concept can improve the quality of reconstructed images and reduce the required time.

In the future, we may design other methods to improve our algorithm, such as reducing the record information. And our codebook is a local codebook, so it may be extended to the global codebook including the block orientation. Thus, we can get a better compression ratio and the high performance.

References

[1] C. K. Chan and C. K. Ma, "A fast method of design better codebooks for image vector quantization," *IEEE Transactions on Communications*, Vol. 42, No. 2/3/4, pp. 237–243, Feb./Mar./Apr. 1994.

[2] C. W. Chao, C. H. Hsieh, and P. C. Lu, "Image compression using modified block truncation coding algorithm," *Signal Processing: Image Communication*, Vol. 12, No. 1, pp. 1–11, Mar. 1998.

[3] E. J. Delp and O. R. Mitchell, "Image compression using block truncation coding," *IEEE Transactions on Communications*, Vol. 27, No. 9, pp. 1335–1342, Sep. 1979.

[4] A. Gersho and R. M. Gray, *Vector quantization and signal compression.* Boston, American: Kluwer Academic Publishers, second ed., 1992.

[5] M. C. Huang and C. B. Yang, "Fast algorithm for designing better codebooks in image vector quantization," Vol. 36, pp. 3265–3271, Dec. 1997.

[6] A. E. Jacquin, "Image coding based on a fractal theory of iterated contractive image transformations," *IEEE Transactions on Image Processing*, Vol. 1, No. 1, pp. 18–30, Jan. 1992.

[7] A. E. Jacquin, "Fractal image coding: a review," *Proceedings of the IEEE*, Vol. 81, No. 10, pp. 1451–1465, Oct. 1993.

[8] C. H. Kuo and C. F. Chen, "A vector quantization scheme using prequantizers of human visual effects," *Signal Processing: Image Communication*, Vol. 12, No. 1, pp. 13–21, Mar. 1998.

[9] C. H. Lee and L. H. Chen, "A fast search algorithm for vector quantization using mean pyramids of codewords," *IEEE Transactions on Communications*, Vol. 43, No. 2/3/4, pp. 1697–1702, Feb./Mar./Apr. 1995.

[10] Y. Linde, A. Buzo, and R. M. Gray, "An algorithm for vector quantizer design," *IEEE Transactions on Communications*, Vol. C-28, No. 1, pp. 84–95, Jan. 1980.

[11] C. K. Ma and C. K. Chan, "Maximum descent method for image vector quantization," *Electronics Letters*, Vol. 27, No. 12, pp. 1772–1773, Sep. 1991.

[12] K. Masselos, T. Stouraitis, and C. E. Goutis, "Novel codebook generation algorithms for vector quantization image compression," *Proceedings of the IEEE ICASSP 98*, Vol. 5, pp. 2661–2664, 1998.

[13] S. A. Mohamed and M. M. Fahmy, "Image compression using VQ-BTC," *IEEE Transactions on Communications*, Vol. 43, No. 7, pp. 2177–2182, July 1995.

Table 1: Analysis of Algorithm Orientation-Partition. Image size: 512×512, codebook size: 256, bpp: 0.8125.

(a)Partition iteration and encoded image performance.

Partition iteration	PSNR			
	Lena	Baboon	Pepper	F16
1	30.864	23.180	29.061	32.032
2	31.134	23.244	30.312	32.282
3	31.195	23.425	31.083	32.327
4	31.211	23.403	31.132	32.332
5	31.229	23.404	31.168	32.402
6	31.201	23.437	31.167	32.388
7	31.232	23.480	31.200	32.412
8	31.256	23.502	31.211	32.415
9	31.254	23.515	31.213	32.429
10	31.254	23.514	31.214	32.429

(b)Execution time and encoded image performance.

Partition iteration	Lena		Pepper	
	PSNR	Execution time(sec)	PSNR	Execution time(sec)
1	30.864	3.536	29.061	3.519
2	31.134	4.471	30.312	4.359
3	31.195	5.386	31.083	5.205
4	31.211	6.355	31.132	6.160
5	31.229	7.274	31.168	7.090
6	31.201	8.201	31.167	8.029
7	31.232	9.080	31.200	8.815
8	31.256	9.735	31.211	9.421
9	31.254	9.965	31.213	9.670
10	31.254	9.995	31.214	9.685

Table 2: Performance of our algorithms. Image size: 512×512, codebook size: 256, bpp: 0.8125.

(a)Performance of Algorithm A.

Image	PSNR	MSE	Execution time(sec)
Lena	31.083	50.672	4.424
Baboon	23.251	307.603	4.306
Pepper	29.061	80.717	3.381
F16	32.191	39.266	4.639

(b)Performance of Algorithm B.

Image	PSNR	MSE	Execution time(sec)
Lena	31.161	49.778	5.311
Baboon	23.390	297.903	5.205
Pepper	31.142	49.995	5.551
F16	32.342	37.918	5.629

Table 3: Performance of the conventional VQ using the MD algorithm. Image size: 512×512. codebook size: 512, codeword size: 4×4, bpp: 0.8125.

			Time (sec)		
Image	PSNR	MSE	Codebook generation	Encoding	Total
Lena	30.861	53.329	2.940	13.964	16.904
Baboon	23.291	304.778	2.720	13.960	16.680
Pepper	30.643	56.081	2.860	13.950	16.810
F16	31.978	41.233	2.855	13.951	16.806

[14] N. M. Nasrabadi and R. A. King, "Image coding using vector quantization: a review," *IEEE Transactions on Communications*, Vol. 36, No. 8, pp. 957–971, Aug. 1988.

[15] K. N. Ngan and H. C. Koh, "Predictive classified vector quantization," *IEEE Transactions on Image Processing*, Vol. 1, No. 3, pp. 269–280, July 1992.

[16] H. B. Park and C. W. Lee, "VQ on projection domain of image," *Signal Processing: Image Communication*, Vol. 5, No. 3, pp. 209–217, May 1993.

[17] B. Ramamurthi and A. Gersho, "Classified vector quantization of images," *IEEE Transactions on Communications*, Vol. 34, No. 11, pp. 1105–1115, Nov. 1986.

[18] S. C. Tai, *Data compression.* Taipei, Taiwan: Unalis, second ed., 1998.

[19] V. R. Udpikar and J. P. Raina, "BTC image coding using vector quantization," *IEEE Transactions on Communications*, Vol. 35, No. 3, pp. 352–355, Mar. 1987.