Image Vector Quantization with Hierarchical Tree Structure[†]

Jian-Ming Chen(陳建明)*,
Chang-Biau Yang(楊昌彪)** and Kuo-Si Huang(黃國璽)**

* Department of Applied Mathematics,
** Department of Computer Science and Engineering,
National Sun Yat-sen University, Kaohsiung, Taiwan
Email:cbyang@cse.nsysu.edu.tw , TEL:886-7-5252000 ext. 4302

Abstract

The tree-structured vector quantization (TSVQ) is an efficient method for the vector quantization (VQ). In this paper, we propose a new tree structure model, hierarchical tree structure (HTS), to reduce encoding time. Briefly, the HTSVQ decreases the dimension of the vector on the upper levels of the codebook tree. If the vector dimension decreases, then the searching time on the codebook tree can be reduced. The experiment results show that the HTSVQ spends less encoding time than the conventional TSVQ and some other fast searching algorithms, and the image quality of HTSVQ is comparable to that of TSVQ.

1. Introduction

The quantization method is one of effective methods for image compression to achieve high compression ratio. The Shannon rate distortion theory shows that better performance is always achievable by encoding vectors instead of scalars [17]. Vector quantization (VQ) [1,14,15] is one effective method, and it is simple to implement.

There are two major issues in VQ. In one efficient scheme, we have to generate a good codebook, and we also need a good encoding scheme to reduce the codeword searching time. Many codebook generating algorithms have been proposed, such as Linde-Buzo-Gray (LBG) [14], pairwise nearest neighbor (PNN) [8], maximum descent (MD) [3]. The LBG algorithm is an iterative method to improve the codebook to a better one. The LBG algorithm generates a good codebook, but its disadvantage is that it needs much time to produce the codebook. The PNN algorithm [8] is very different from the LBG algorithm. The PNN is a bottom-up algorithm. Initially, each vector of the training set is viewed as an individual cluster. Then, two nearest clusters are merged into one cluster repeatedly until N clusters are obtained. The MD algorithm [3] is a top-down algorithm to generate a codebook. The initial cluster is the whole training set. The cluster with the maximum reduction of the distortion is selected to be split repeatedly, until N clusters are got.

In order to partition one cluster into two clusters efficiently, we may have to obtain the splitting axes in a fast way. The longest distance partition (LDP) algorithm is an efficient cluster splitting algorithm [10], which gets the splitting axes in $2n_i$ comparisons. It can be used as a splitting procedure in the MD, the 2-level LBG algorithm, or any other algorithm which needs splitting.

For the codeword searching, full search is the simplest. However, it costs a lot of time. For each vector x of an image, the full search requires N distortion calculations to find the nearest codeword of the codebook. Many efficient methods have been proposed for this problem, such as the tree-structured vector quantization (TSVQ) [2, 11, 19, 20], the partial distortion search method (PDS) [16], the dynamic windowed codebook search (DWCS) algorithm [13], and the triangle inequality eliminating ruler (TER) [9].

The partial distance search (PDS) [16] method reduces the computational complexity for searching the minimum distortion in the encoding phase. The PDS algorithm provides the minimum distortion searching method without computing the distance for some codewords. The triangle eliminating ruler (TER) [9] clusters the codewords satisfying triangle inequality property as a new subcodebook, and we can search the subcodebook to get the nearest codeword. The fast triangle inequality eliminating ruler (FTER) [12] gets a close initial codeword than the TER by decreasing the size of the subcodebook in the beginning of the codebook searching. The mean-distance-ordered partial codebook search (MPS) [18] uses the mean values of the codewords as the indices of the codewords. The idea of the MPS is that the minimum squared Euclidean distance (SED) codeword is usually in the neighborhood of the minimum squared mean distance (SMD). The dynamic windowed codebook search (DWCS) [13] algorithm is associated with a key value, which is obtained by projecting the corresponding codeword onto an analyzed vector u_1. An improved DWCS [13] uses the

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

four neighbor codewords as the initial minimum bound on the second and third principal directions u_2 and u_3. It gets a smaller bound on the second and third projection.

The TSVQ is an efficient method for the VQ encoding. The TSVQ uses the binary search scheme on codeword searching. We can find the codeword from the codebook with *log N* steps, but the codeword found by TSVQ is not really the nearest codeword to the image vector *x*. Some methods are proposed to overcome this disadvantage, such as the multipath TSVQ [5, 6] and the closest-coupled tree-structured vector quantization (CCTSVQ) [4]. The multipath TSVQ increases the number of searching paths to find a close codeword on the codebook tree. The encoding time also increases as the number of searching paths increases. Such ways to find more precise codeword always spend more encoding time.

In this paper, we propose a hierarchical tree structure vector quantization (HTSVQ). The HTSVQ is more efficient then the conventional TSVQ in the VQ encoding phase. The HTSVQ divides the codebook tree into three classes by the levels of the tree. In class A of the tree, we transform each vector into a scalar value. In class B, we transform each vector into a 4-dimensional vector. In class C, we donnot do any change on each vector. The required time for distortion measure decreases as the vector dimension is reduced. In class A and class B, because the vector dimension decreases, we can reduce the required time for distortion measure when we perform search on the tree.

The rest of this paper is organized as follows. Section 2 gives the encoding algorithm based on the hierarchical tree structure. The result of computational experiments and comparison with other VQ-based algorithms are given in Section 3. Finally, Section 4 concludes the paper.

2. The Hierarchical Tree-structured Vector Quantization

In this section, we will describe our hierarchical tree structure. Our idea is to reduce the number of vector dimensions on the upper levels of the codebook tree. Thus, we call it the hierarchical tree structure (HTS). In our scheme, the HTS divides the codebook tree into three classes by the level of the tree. Classes *A*, *B*, and *C* contains α upper levels, β middle levels and γ lower levels on the codebook tree, respectively. In class *A*, each vector is transformed to a scalar value (1-dimensional vector) with function f_1. That is, $f_1 : R^k \rightarrow R$, where *k* is the number of elements in one block. In class *B*, each vector is transformed to a 4-dimensional vector with function f_2. That is, $f_2 : R^k \rightarrow R^4$. And we keep each vector in class *C* unchanged. We use functions f_1 and f_2 to reduce the vector dimension.

The function we use to reduce the vector dimension is the vector sum function. Two close vectors usually have close vector sums. Given a 16-dimensional vector, $X = (x_1, x_2, \ldots, x_{16})$, we define our vector sum functions f_1 and f_2 as follows:

$$f_1(X) = \sum_{i=1}^{16} x_i$$

$f_2(X) = Y = (y_1, y_2, y_3, y_4)$, where

$$y_1 = x_1 + x_2 + x_5 + x_6$$
$$y_2 = x_3 + x_4 + x_7 + x_8$$
$$y_3 = x_9 + x_{10} + x_{13} + x_{14}$$
$$y_4 = x_{11} + x_{12} + x_{16} + x_{16}$$

The distortion measurement is also modified according to the vector dimension.

There are two viewpoints that we use the vector sum function to reduce the vector dimension. First, the vector sum function projects each vector onto the (1, 1, …, 1) axes. And the splitting axes of class *A* is $(b_1, b_2, \ldots, b_K)$, where $b_I = 1$, $1 \le i \le K$. And the vector sum function can be easily calculated. On the middle or lower levels of the tree, we need more splitting axes to get better splitting effect. Thus, a vector is divided into four parts, and the sum of each part is calculated. Thus, that is 4-dimensional vectors are used. In class *C*, each vector remains unchanged.

The other viewpoint is that the vector sum function can reduce the time required for searching a close vector. Although two closest vector sums do not mean that the two vectors are the closest, but the two closest vectors usually have close vector sums. Searching the codeword with its vector sum can remove most impossible codewords. On the middle or lower levels of the tree, the difference between two vector sums decreases, and the estimation error increases. We need more accurate estimation. Thus, we divide the vectors into four disjoint regions, and calculate the vector sum of each region.

Now, we shall present our HTS algorithm more precisely. As shown in Figure 1, we divide a codebook tree into three classes *A*, *B*, and *C* by the level of the codebook tree, denoted as $H(\alpha,\beta,\gamma)$. In $H(\alpha,\beta,\gamma)$, class *A* consists of 1-dimensional vectors, and the height (depth) of the sub-tree is α. Class *B* consists of 4-dimensional vectors and the height of the sub-tree is β. Class *C* consists of *K*-dimensional vectors and its height is γ. Classes *A*, *B*, *C* are on the upper, middle and lower levels of the codebook tree, respectively. After class *A* is built completely, we take each leaf node on the codebook tree as one root of class *B*, and built the subtrees for class *B*. Similarly, each leaf node of the class *B* is one root in class *C*. The distortion measurement on each node of class *A* and class *B* applies f_1 and f_2 first, respectively. Our HTS codebook tree generation algorithm is as follows.

Algorithm: HTSVQ Codebook Tree Generation

Input: A training set $TS = \{x_1, x_2, \ldots, x_m\}$. α, β, γ and threshold δ.

Output: A $H(\alpha,\beta,\gamma)$ codebook tree with $2N$ leaves, where $N = 2^{\alpha+\beta+\gamma}$.

Step 1: Let *class* = *A* and height $h = \alpha$.

Step 2: According the value of class, perform the corresponding case as follows.

Case 2.1 ***class*** **is** ***A*:** Set each cluster C_i, $1 \le i \le 2N$, to be empty. Calculate the centroid of C_{f_1}, where $C_{f_1} = \{f_1(x_1), f_1(x_2), \ldots, f_1(x_m)\}$. Let cluster $TC = C_{f_1}$. And let $j = 1$. Calculate the centroid of TC as the root of the HT.

Case 2.2 ***class*** **is** ***B*:** Transform the C_i cluster, for all $f_1(x) \in C_i$, $C_i = C_i \cup \{f_2(x)\} - \{f_1(x)\}$, for $i = 2^{\alpha}, 2^{\alpha}+1, \ldots, 2^{\alpha+1}-1$. Let $j = 2^{\alpha}$, $TC = C_{2^{\alpha}}$.

Case 2.3 ***class*** **is** ***C*:** Transform each vector in cluster C_i. That is, for all $f_2(x) \in C_i$, $C_i = C_i \cup \{x\} - \{f_2(x)\}$, for $i = 2^{\alpha+\beta}, 2^{\alpha+\beta}+1, \ldots, 2^{\alpha+\beta+1}-1$. Let $j = 2^{\alpha+\beta}$, $TC = C_{2^{\alpha+\beta}}$.

Step 3: Apply the LDP algorithm to generate v_l, v_r by TC.

Step 4: Apply the LBG algorithm to split TC into two clusters C_l, C_r. And calculate their centroids u_l, u_r.

Step 5: Let $C_{2j} = C_l$, $C_{2j+1} = C_r$. Let u_l and u_r be the left son and right son of node j.

Step 6: If $j < 2^{h+1}$, set $TC = C_{j+1}$, $j = j+1$, and go to Step 3.

Step 7: Perform one of the following cases.

Case 7.1 ***class*** **is** ***A*:** Let $class = B$, height $h = \alpha+\beta$, and goto step 2.

Case 7.2 ***class*** **is** ***B*:** Let $class = C$, height $h = \alpha+\beta+\gamma$, and goto step 2.

Case 7.3 ***class*** **is** ***C*:** Stop.

When the codebook tree is built, the vectors associated with each node are split into two clusters. And the vectors in these two clusters are associated with the two sons of that node, respectively. When the splitting is performed, the LDP algorithm [10] is applied. The LDP algorithm is efficient for partitioning one cluster into two clusters.

Since the number of dimensions of each vector on the upper and middle levels of the tree is reduced to one and four, respectively, the distance computation time is reduced. Thus, the codebook generation with HTS is faster then the conventional TSVQ.

Besides, in the encoding phase, searching a close representative codeword for each vector also follows the concept of reduced dimension. Thus, our algorithm also provides a fast searching scheme.

3. Experiment Results and Performance Analysis

In this section, we shall illustrate the performance of our encoding algorithm with the HTS. We test the HTS both on the local codebook tree and the global codebook tree. Coding an image with a local codebook has better quality than coding with a global codebook. However, there are two disadvantages in coding with the local codebook. The first one is that the codebook generation spends much time. Generating a codebook for all images saves much time. We can generate a global codebook for all images. We build a codebook in the first time when we need it, and we can store it on the computer. When we encodes other images, we can use it again. The other disadvantage of the local codebook scheme is that each image has to allocate one individual codebook for encoding, which requires large storage space. However, image coding with global codebook may not get a reconstructed image with good quality when the image is not similar to the training images for the global codebook. When we want to get an image coding method, we can use the global codebook scheme. And if we want to get good image quality, we should use a local codebook. We will present the performance of the HTS coding on both the local codebook and the global codebook.

We establish one local codebook for each image of Lena, Pepper and Baboon. We use Lena, Pepper, Baboon, Bridge and Bear as the global codebook training set. All images are standard monochrome pictures, each containing 512 × 512 pixels. Each pixel on the pictures has 256 gray levels. We decompose an image into a set of blocks with size 4 × 4, which are viewed as 16-dimensional vectors. Our simulation are preformed on the IBM compatible PC with Intel Celeron 333(Pentium II compatible) and 64MB RAM. The quality of the reconstructed image with VQ is measured by the peak signal-to-noise (PSNR), which is defined as follows:

$$PSNR = 10 \log_{10} \frac{255^2}{MSE} dB,$$

where the mean square error (MSE) for a $L \times L$ image is defined as

$$MSE = \frac{1}{L^2} \sum_{i=1}^{L} \sum_{j=1}^{L} (w_{ij} - \hat{w}_{ij})^2,$$

where $L \times L$ is the size of image, w_{ij} is the pixel value of the original picture at coordinate (i, j), and $\hat{w}_{ij}$ is the pixel value of the reconstructed picture at coordinate (i, j).

Figure 2 shows the reconstructed image of Lena with the H(2,2,4) on the local codebook. Table 1 shows the encoding time and the quality of the reconstructed image of the HTS and the conventional TSVQ on the local codebook with size 256, respectively. The testing image of Table 1(a) is Lena. The tree searching time on the H(2,2,4) codebook tree is only 58.7% of that on the conventional TSVQ, and the encoding image quality of the H(2,2,4) is comparable to that of the conventional TSVQ. The encoding time on H(2,3,3) is 50.7% of that on the conventional TSVQ. The encoding time on H(2,2,4) is only 4.1% of that of the full search scheme. Table 1(b) is the result for image Pepper.

Table 2 shows the performance on the global codebook with size 256. The training images are Lena, Pepper, Baboon, Bridge and Bear. Table 2(a) shows the time required for the conventional TSVQ, H(2,2,4), H(2,3,3) and H(2,4,2). The codebook generating time required for H(2,2,4) is 74.6% of that of the conventional TSVQ. The tree-structured search time is the same as that in Table 2(a). For the global codebook scheme, Table 2(b) shows the quality of the reconstructed image on TSVQ, H(2,2,4), H(2,3,3) and H(2,4,2), respectively. Table 3 shows the performance the global codebook with size 512.

Table 4 shows the performance of some fast searching algorithms on the global codebook with size 256 and 512. The training images are Lena, Pepper, Baboon, Bridge and Bear. Table 4(a) and 4(b) shows the required time and the PSNR for Lena, and Baboon, respectively. The quality of reconstructed images of our algorithm is comparable to that of other algorithm. And our algorithm still needs less searching time (encoding time).

4. Conclusion

In this paper, we propose a hierarchical tree structure vector quantization (HTSVQ). The HTSVQ is more efficient then the conventional TSVQ in the VQ encoding phase. The HTSVQ divides the codebook tree into three classes by the levels of the tree. In class *A* of the tree, we transform each vector into a scalar value. In class *B*, we transform each vector into a 4-dimensional vector. In class *C*, we donnot do any change on each vector. The required time for distortion measure decreases as the vector dimension is reduced. In class *A* and class *B*, because the vector dimension decreases, we can reduce the required time for distortion measure when we perform search on the tree.

In our experiment results, the HTSVQ reduces about 40% encoding time than the conventional TSVQ, and the image quality is comparable to that of TSVQ. It also spends less encoding time then other fast search algorithms.

There are many other functions that can be used on the vector dimension reduction function, such as projecting the vector on the discrete cosine transform (DCT) base or the Walsh-Hadamard transform (WHT) base. In the future, it may be worth to study the combination of HTS and other reduction functions.

Reference

[1] C. D. Bei and R. M. Gray, "An improvement of the minimum distortion encoding algorithm for vector quantization", IEEE Transactions on Communications, Vol. C?3, No. 10, pp. 1132--1133, Oct. 1985.

[2] A. Buzo, A. H. Gray, R. M. G. Jr, and J. D. Markel, "Speech coding based on vector quantization", IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. ASSP?8, No. 10, pp. 562--574, Oct. 1980.

[3] 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.

[4] C. C. Chang and T. S. Chen, "New tree-structured vector quantization with closest-coupled multipath searching method", Optical Engineering, Vol. 36, No. 6, pp. 1713--1720, June 1997.

[5] F. Chang, W. T. Chen, and J. S. Wang, "Image sequence coding using adaptive tree-structre vector quantization with multipath searching", IEEE Int. Conf. Acoust., Speech, and Signal Processing, pp. 2281--2284, 1991.

[6] R. F. Chang., W. T. Chen, and J. S. Wang, "Image sequence coding adaptive tree-structured vector quantization with multipath searching", IEE Proceedings?, Vol. 139, No. 1, pp. 9--4, Jan. 1992.

[7] W. H. Cooley and P. R. Lohnes, Multivariate data analysis. New York: Wiley, 1971.

[8] W. H. Equitz, "A new vector quantization clustering algorithm", IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. 37, No. 10, pp. 1568--1575, Oct. 1989.

[9] C. M. Huang, Q. Bi, G. S. Stiles, and R. W. Harris, "Fast full search equivalent encoding algorithm for image compression using vector quantization", IEEE Transcations on Image Process, No. 1, pp. 413--416, Jan. 1992.

[10] M. C. Huang and C. B. Yang, "Fast algorithm for designing better codebooks in image vector quantization", Optical Engineering, Vol. 36, No. 12, pp. 3265--3271, Dec. 1997.

[11] 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.

[12] Y. C. Lin and S. C. Tai, "Fast feature-based vector quantization algorithm of image coding", Optical Engineering, Vol. 34, No. 10, pp. 2918--2926, Oct. 1995.

[13] Y. C. Lin and S. C. Tai, "Dynamic windowed codebook search algorithm in vector quantization", Optical Engineering, Vol. 35, No. 10, pp. 2921--2929, Oct. 1996.

[14] Y. Linde, A. Buzo, and R. M. Gray, "An algorithm for vector quantizer design", IEEE Transactions on Communications, Vol. C?8, No. 1, pp. 84--95, Jan. 1980.

[15] 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.

[16] K. K. Paliwal and V. Ramasubramanian, "Efficient of ordering the codebook on the efficiency of the partial distance search algorithm for vector quantization", IEEE Transactions on Communications, Vol. 37, No. 11, pp. 538--540, Nov. 1989.

[17] W. K. Pratt, Digital image processing. New York,

American: John Wiley & Sons, second ed., 1991.

[18] S. W. Ra and J. K. Kim, "A fast mean-distance-ordered partial codebook search algorithm for image coding", IEEE Transcations on Circuits and Systems-II Analog and Digital Signal Processing, Vol. 40, No. 9, pp. 576--579, Sep. 1993.

[19] V. Ramasubramanian and K. K. Paliwal, "Fast k-dimensional tree algorithm for nearest neighbor search with application to vector quantization encoding", IEEE Transactions on Signal Processing, Vol. 40, No. 3, pp. 518--531, Mar. 1992.

[20] V. S. Sitaram, C. M. Huang, and P. D. Israelsen, "Efficient codebooks for vector quantization image compression with an adaptive tree search algorithm", IEEE Transactions on Communications, Vol. 42, No. 11, pp. 3027--3033, Nov. 1994.

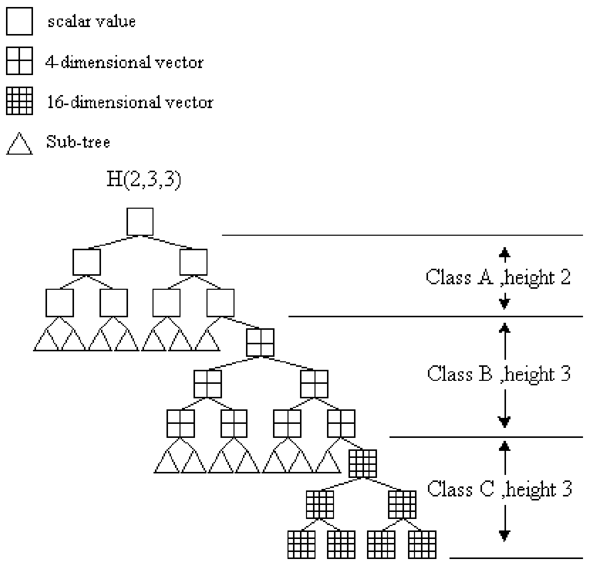

Figure 1: An example H(2,3,3) of the hierarchical tree-structured codebook model.

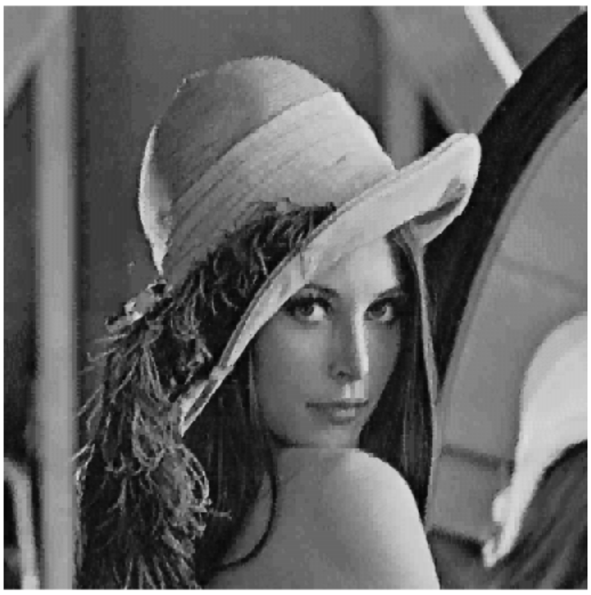

Figure 2: The reconstructed image of Lena with the H(2,2,4) on the local codebook. Image size: 512 × 512, gray level: 256, vector size: 4 × 4, codebook size: 256, bpp: 0.625, PSNR: 29.6274.

Table 1: The performance of the HTSVQ with the local codebook. Image size: 512 × 512, codebook size: 256. TS: conventional tree structure scheme. H(α,β,γ): hierarchical tree structure scheme.

(a) The performance for Lena.

Image	Lena 512×512			
Tree structure	TS	H(2,2,4)	H(2,3,3)	H(2,4,2)
Codebook generating time(sec)	1.9630	1.4550	1.3650	1.2840
Tree search encoding time(sec)	0.2640	0.1550	0.1340	0.1110
Total time(sec)	2.2270	1.6100	1.4990	1.3950
TS search MSE	70.8172	70.8508	71.3634	71.8700
TS search PSNR	29.6294	29.6274	29.5960	29.5653
Full search encoding time(sec)	3.7850	3.7290	3.9850	3.9540
Full search MSE	64.5526	64.5317	64.4287	65.2593
Full search PSNR	30.0317	30.0331	30.0400	29.9844

(b) The performance for Pepper.

Image	Lena 512×512			
Tree structure	TS	H(2,2,4)	H(2,3,3)	H(2,4,2)
Codebook generating time(sec)	1.8240	1.4050	1.3160	1.2350
Tree search encoding time(sec)	0.2860	0.1540	0.1350	0.1110
Total time(sec)	2.1100	1.5590	1.4510	1.3460
TS search MSE	76.1875	77.6647	77.1985	77.7180
TS search PSNR	29.3120	29.2286	29.6568	29.2256
Full search encoding time(sec)	3.8600	3.8990	3.8800	3.9040
Full search MSE	69.2943	70.2557	70.0362	70.3716
Full search PSNR	29.7238	29.6640	29.6776	29.6568

Table 2: The performance of the HTSVQ with the global codebook. Training images: Lena, Pepper, Baboon, Bridge and Bear; image size: 512 × 512, codebook size: 256. TS: conventional tree structure scheme. H(α,β,γ): hierarchical tree structure scheme.

(a) The time required for the TSVQ and various HTSVQs.

Tree structure	TS	H(2,2,4)	H(2,3,3)	H(2,4,2)
Codebook generating time(sec)	10.510	7.845	7.346	6.900
Tree-structured encoding time(sec)	0.260	0.160	0.135	0.115
Full search encoding time(sec)	3.765			

(b) The PSNRs for the TSVQ and various HTSVQs.

Codebook structure		TS	H(2,2,4)	H(2,3,3)	H(2,4,2)
Image	Encoding method	PSNR			
Lena	Tree search	28.7993	28.7181	28.6649	28.6058
	Full search	29.1467	29.0602	28.9996	28.9581
Pepper	Tree search	28.6780	28.7008	28.6365	28.6775
	Full search	29.0292	29.0751	29.0307	29.0672
Baboon	Tree search	21.8896	21.8986	21.8925	21.8646
	Full search	22.1338	22.1518	22.1545	22.1370
Bridge	Tree search	24.6706	24.6574	24.6247	24.6177
	Full search	24.9453	24.9278	24.9181	24.9169
Bear	Tree search	28.6321	28.6182	28.6399	28.6196
	Full search	29.0225	29.0221	29.0201	29.0126

Table 3: The performance of the HTSVQ with the global codebook. Training images: Lena, Pepper, Baboon, Bridge and Bear; image size: 512 × 512, codebook size: 512. TS: conventional tree structure scheme. H(α,β,γ): hierarchical tree structure scheme.

(a) The time required for the TSVQ and various HTSVQs.

Tree structure	TS	H(2,2,4)	H(2,3,3)	H(2,4,2)
Codebook generating time(sec)	12.590	10.064	9.430	8.960
Tree-structured encoding time(sec)	0.295	0.195	0.170	0.145
Full search encoding time(sec)	7.845			

(b) The PSNRs for the TSVQ and various HTSVQs.
Codebook structure TS H(2,2,5) H(2,3,4) H(2,4,3)

Codebook structure		TS	H(2,2,4)	H(2,3,3)	H(2,4,2)
Image	Encoding method	PSNR			
Lena	Tree search	29.4468	29.3963	29.3594	29.3355
	Full search	29.8607	29.7937	29.7671	29.7491
Pepper	Tree search	29.5032	29.5168	29.4131	29.4212
	Full search	29.8907	29.9079	29.8307	29.8333
Baboon	Tree search	22.3486	22.3604	22.3614	22.3395
	Full search	22.6884	22.6940	22.7240	22.6949
Bridge	Tree search	25.1544	25.1728	25.1378	25.1311
	Full search	25.5081	25.4915	25.4751	25.4547
Bear	Tree search	29.2242	29.2481	29.2394	29.2384
	Full search	29.6837	29.6905	29.6857	29.6834

Table 4: The performance of several fast search algorithms with the global codebook. Training images: Lena, Pepper, Baboon, Bridge and Bear. image size: 512 × 512, codebook size: 512 and 256. Note: FSVQ(Full Search VQ), DWCS(PCA [7]) and DWCS(DCT) [13]

(a) The performance for Lena.

Image	Lena 512 × 512			
Codebook size	256		512	
Searching algorithm	Encoding time	PSNR	Encoding time	PSNR
FSVQ	3.76000	29.0602	8.64900	29.7491
TSVQ	0.26000	28.7993	0.29500	29.4468
HTSVQ-H(2,2,4)	0.16000	28.7181		
HTSVQ-H(2,3,4)			0.17000	29.3594
TER	1.39000	29.0602	2.95400	29.7491
FTER	0.43500	29.0602	0.79600	29.7491
MPS	0.22000	29.0602	0.34000	29.7491
DWCS(PCA)	0.39500	29.0602	0.46500	29.7491
DWCS(DCT)	0.38000	29.0602	0.53500	29.7491

(b) The performance for Baboon.

Image	Baboon 512 × 512			
Codebook size	256		512	
Searching algorithm	Encoding time	PSNR	Encoding time	PSNR
FSVQ	3.76000	22.1518	8.64900	22.7240
TSVQ	0.26000	21.8646	0.29500	22.3486
HTSVQ-H(2,2,4)	0.16000	21.8986		
HTSVQ-H(2,3,4)			0.17000	22.3614
TER	2.27000	22.1518	4.57500	22.7240
FTER	0.99400	22.1518	1.97000	22.7240
MPS	0.54000	22.1518	0.91500	22.7240
DWCS(PCA)	0.85000	22.1518	1.29100	22.7240
DWCS(DCT)	0.79000	22.1518	1.31000	22.7240