h HKUST 香港科技大學

President's Cup

Application of Real time Searching Algorithm in Image encoder

Personal Details

School	**Engineering**
Department	**EEIC**
Group Member 1	
Name:	**Lam Chi Wai Lance**
Student ID	
Email	
Group Member 2	
Name:	**Chau Chun Mei**
Student ID	
Email	
Group Member 3	
Name:	**Tam Ka Yan**
Student ID	
Email	
Project Supervisor	**Dr. Kam Tim, Woo (EE Department)**
Project Supervisor's Signature	
Date	**13-4-07**

Table of Content

List of Illustrations

Figure

TABLE

Chapter 1 Abstract

An image or video transmission becomes a huge demand in the nowadays applications. Vector Quantization (VQ) has become established as a low-bit rate data compression technique by classifying the similar patterns into a finite number of groups. However, its utility for real-time coding applications has been quite limited due to its high encoding complexity.

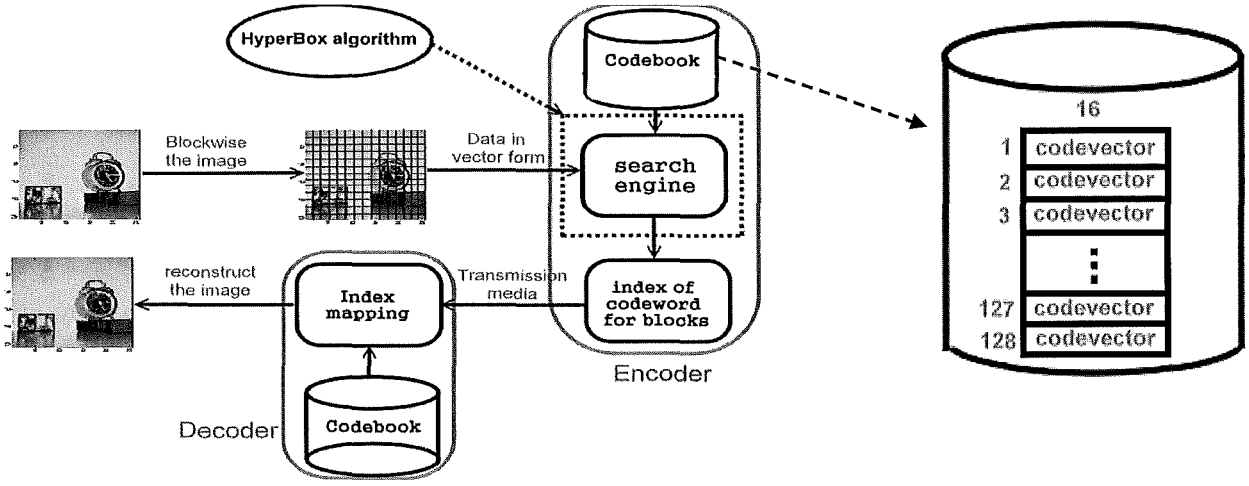


Figure 1 flow of image reconstruction

Figure 1 shows the basic block diagram of the vector quantization technique on image compression. Firstly, the original image was sub-divided into the blockwise images. Each blockwise image was then encoded as an index of the nearest neighbor codevector that provides the smallest Euclidean distance between the input blockwise image and each codevector in the codebook. The index of this codevector was sent through a media such as communications channel, computer storage. At the decoder, the image was reconstructed by assigning the codevector associated with the received index.

The Search engine in the encoder process

Assuming a particular codebook has been specified, we focused on the design of an efficient search engine in the encoder. The search engine is enclosed by the dotted lines in **Figure 1**. We proposed a new search algorithm named **HyperBox algorithm**. The algorithm aims to lower the computation complexity without scarifying image quality.

Chapter 2 Background Information

For real-time video coding applications, the processes of quantizing the input vectors are required to be fast. Since the nearest codevector search is involved in the encoding processes, a fast codevector search algorithm can make the coding process time efficient.

2.1 Introduction of some searching algorithms

2.1.1 Full Search

The basic idea to search the most representative codevector from codebook is to consider all codevectors for distance measure. The codevector with the minimum distance to a query vector will be the most suitable one for encoding to achieve the least distortion. As it is known that the larger the set of codevectors M is, the more accurate vector quantization (VQ). However, when M is large, the computational complexity problem for full codebook search will occur. This problem is critical in the codebook design and encoding of **VQ.** To avoid such an exhaustive search through the codebook, many fast algorithms have been proposed. These algorithms reduce the computational complexity by first performing some simple tests before computing the distortion between the training vector and each codeword, and then rejecting those codewords which fail in the tests.

2.1.2 Equal-average Nearest Neighbor Search (ENNS), (Modified) ENNS, (Second Modified)ENNS

These three algorithms, which are **some existing fast closest codeword search algorithms,** commonly use the mean value and the sum of the absolute differences as the two criteria to reject unlikely codevectors, thereby saving a great deal of computational time, while introducing no more distortion than the conventional full search algorithm. More details can be read in these papers [1][2][3] in the reference section. A brief explanation will provided as follows:

2.1.2.1 Brief Description of ENNS, MENNS, SMENNS

In **figure 2, dmin** is defined as the distance between the closest codevector and the query vector and

$$\sqrt{\sum^{k}\left(\frac{\text{dmin}}{\sqrt{k}}\right)^2} = \text{dmin}$$

where k is the dimension of vector

ENNS – only relies on the mean boundary to find the testing point to do distance evaluation.

MENNS- relies on both the mean boundary and the variance to find the testing point to do distance evaluation.

SMENN - relies on both the mean boundary and the sum of absolute difference to find the testing point to do distance evaluation.

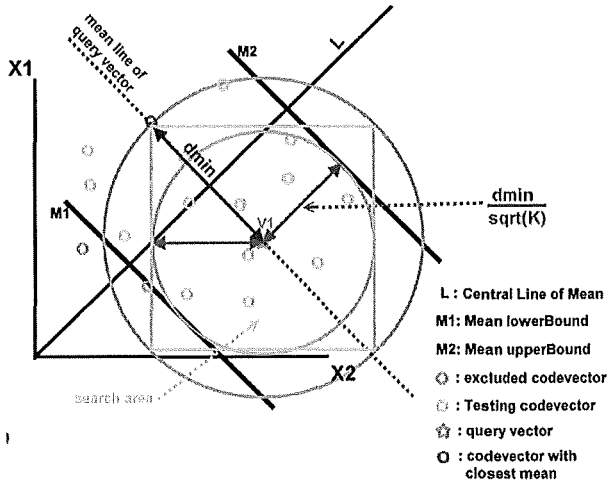


Figure 2 Example of ENNS algorithm for 2-dimensional case

There are two significant points which occupied most of the computation complexity.

1. Search points – The testing points required to perform partial distance evaluation
2. Update points –The hit points required to perform calculation of new mean boundary or other restriction boundary.

With these two types of points, it is straightforward and reasonable to estimate the computation complexity.

2.1.2.2 Weakness of these three methods

The common problem of these three algorithms comes out from the fundamental concept of using the codevector with closest mean as the reference of generating the mean boundary. In **figure 2**, the codevector with closest mean to the query vector seems to be too far away. It was chosen to create the mean boundary (M1 and M2) which generated a larger search area which then implied more search points inside. Although MENNS and SMENNS algorithms try to use the variance and sum of absolute difference to reduce the size of search area by searching for closer codevector so as to narrow the mean boundary, additional memory or computation was required. Also owing to a very low hit rate for initial codevector, that is the codevector with closest mean that still can have a chance to enlarge the search area, use of more search and update points were unavoidable and the calculation involved was the burden to the speed of searching.

Here are some measurements which are used to evaluate the performance of testing algorithm.

2.2 Speed up factor

Speed up factor evaluates the performance of a testing search algorithm based on its computation complexity. It is the ratio of the number of operations in Full Search to that in a testing algorithm. The greater the factor, the performance will sound more attractive.

$$\text{Speed up factor} = \frac{\text{no. of operations in FULL SEARCH}}{\text{no. of operations in TESTING ALOGRITHM}}$$

2.3 PSNR

In the application of clustering on the image, PSNR is commonly used to evaluate the quality of image after reconstruction.

PSNR (in dB) is an objective ratio and a peak signal to noise ratio that is used to evaluate how much the image signal is affected by noise after reconstruction.

$$\text{PSNR (dB)} = 10 \times \log\left(\frac{(2^n - 1)^2}{\text{MSE}}\right)$$

Where n = maximum bits for one pixel

$(2^n - 1)^2$ = maximum signal power of one pixel

2.4 PSNR ratio

In image encoding part, the purpose of PSNR ratio is to evaluate the performance of PSNR using testing algorithm through comparing with that using Full Search. The ratio formulae is shown as follow,

$$\text{PSNR ratio} = \frac{\text{PSNR (in W) by testing algorithm}}{\text{PSNR (in W) by FULL SEARCH}}$$

Chapter 3 Detailed description of the project

3.1 HyperBox algorithm

Motivated by the ENNS algorithm, we made full use of the mean-clustering concept and propose **HyperBox algorithm** to have a fast search from a huge set of codevectors. Before the search algorithm was introduced, some preliminary tasks were needed in the codebook design.

3.1.1 Preliminary tasks in the codebook design:

Step 1

In figure 3.1, the Modified Kmean clustering algorithm was used to cluster a set of training vectors and generate a number of representative codevectors (the purple and green circles) which are less sensitive to initial guesses.

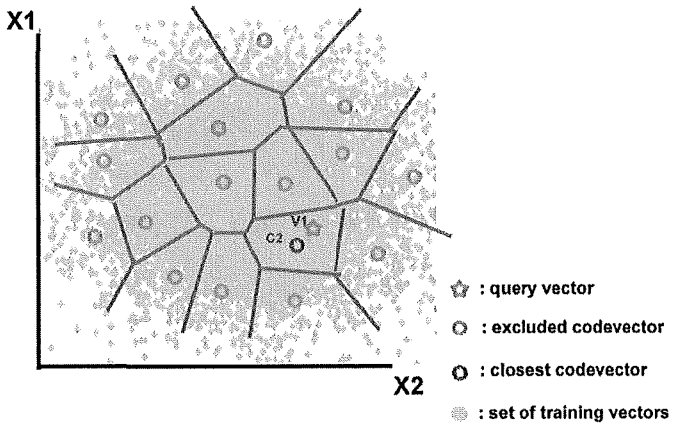


Figure 3.1 codevectors are generated by Modified Kmean Clustering Algorithm

Step 2:

For the purpose of the fast search of the codevectors generated in Step 1, these codevectors were clustered by using Binary Tree clustering algorithm (**figure 3.2**) so that they can be searched by tracing from the tree nodes. (**figure 3.3**)

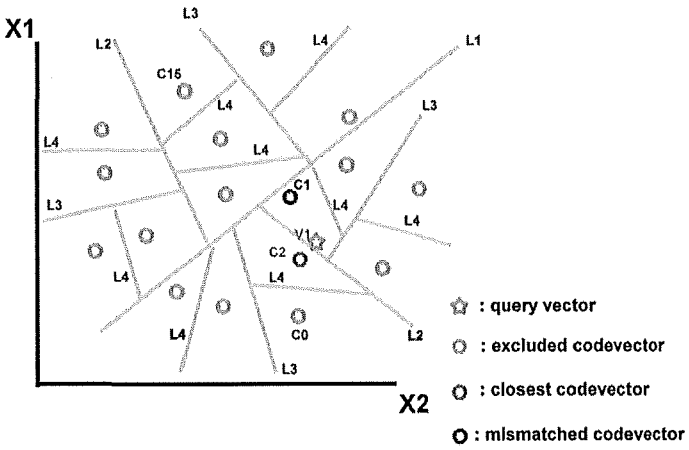


Figure 3.2 The hyper-planes (pink lines) are created by binary tree algorithm

Step 3:

The mean value of each codevectors was pre-computed and was arranged in ascending order and stored in the codebook as well as the final codevectors. Also, pre-computed values were stored in the codebook for the purpose of fast search.

3.1.2 Implementation of proposed algorithm in search engine

Step 1:

The codevectors, mean value and Pre-computed values stored in Codebook were loaded.

Step 2:

The binary tree-based search was used to search the closest codevector to query vector V1 (**Figure 3.3**)

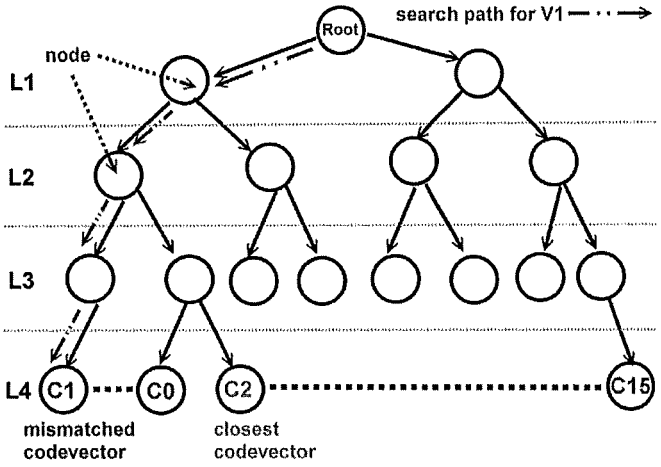


Figure 3.3 Binary tree search of closest codevector and mismatch problem emerges

Figure 3.3 showed the mismatch condition which happened due to the hyper-plane cutting (**figure 3.2**) created by the binary tree clustering algorithm. Although C_1 is very close to query vector, C_2 is the closest one and should be chosen in order to achieve better VQ. So the following steps will adopt the "mean clustering" and "hyperbox" concept to minimize the computations to find the closest codevectors.

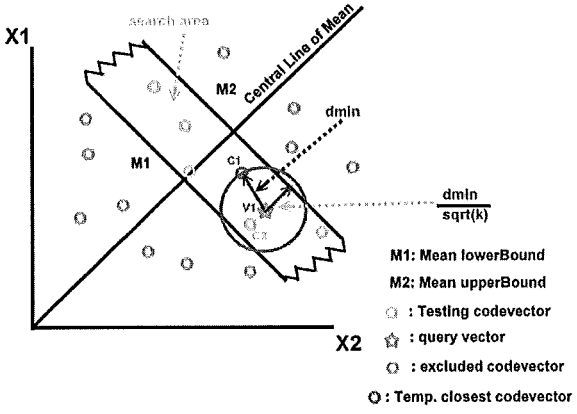


Figure 3.4 The creation of mean boundary

Step 3

In **figure 3.4**, the purpose of the central line of mean is to let those multi-dimension codevectors project to that line so that codevectors with same mean value are grouped together. The search area was bound by the mean boundary (M1 and M2) and only the codevectors with mean value inside this boundary were considered. The calculation of mean boundaries on the central line is shown as follows:

$$\text{M1} = \text{mean upperbound} = \frac{1}{k}\sum_{i=1}^{k} V1_i + \frac{d\min}{\sqrt{k}}$$

$$\text{M2} = \text{mean lowerbound} = \frac{1}{k}\sum_{i=1}^{k} V1_i - \frac{d\min}{\sqrt{k}}$$ where k is dimension of vector

As the codevectors were sorted in ascending order by their mean values, it was easier to use binary search to find the index of codevectors with the mean closest to the upperbound and lowerbound. The purpose of the index search is to fasten the search by considering the range of index inside the boundary instead of using the boolean checking of whether a testing codevector is inside the mean boundary.

Step 4

After Step 3, a number of far-away codevectors are rejected. In order to reduce the search points for distance evaluation, it was proposed to create a boundary for each orthogonal basis (X1, X2) by using upper and lower bounds. These orthogonal bounds checks whether all orthogonal values of a testing codevector are inside the approximated decision region (**figure 3.5**). If one orthogonal value exceeds the orthogonal boundary, the testing codevector will be rejected. This boundary seems like a box and is so called "Hyperbox". The orthogonal upperbound andlower bound can be calculated by following equation:

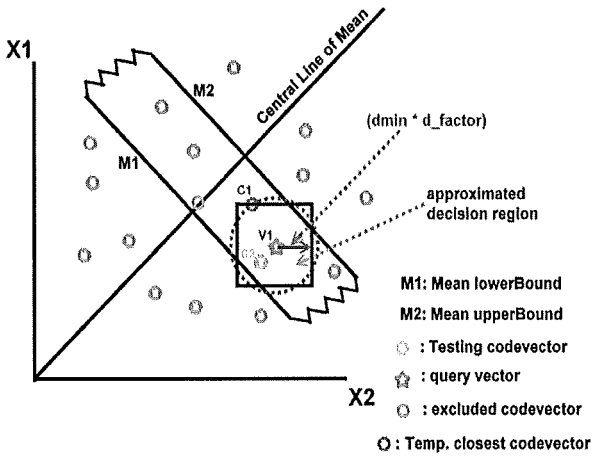


Figure 3.5 The approximated decision region inside the "Hyperbox"

$$\text{hyperbox orthogonal upperbound} = V1_i + \text{dmin} * \text{d\_factor}$$
$$\text{hyperbox orthogonal lowerbound} = V1_i - \text{dmin} * \text{d\_factor}$$
$$\text{where } i = 1,2,3,...k \text{ and k is dimension of vector}$$

Step 5

After step 4, some of the search points were further excluded. Only a small number of search points inside the "hyperbox" were considered in Euclidean distance evaluation.

Step 6

If one codevector has shorter distance than the initial codevector, then that codevector will be regarded as the update point. Afterwards, the update process will launch. The update process includes the calculation of upper-bound of mean as well as the calculation of the "hyperbox" boundary. This process will further reduce the search area.

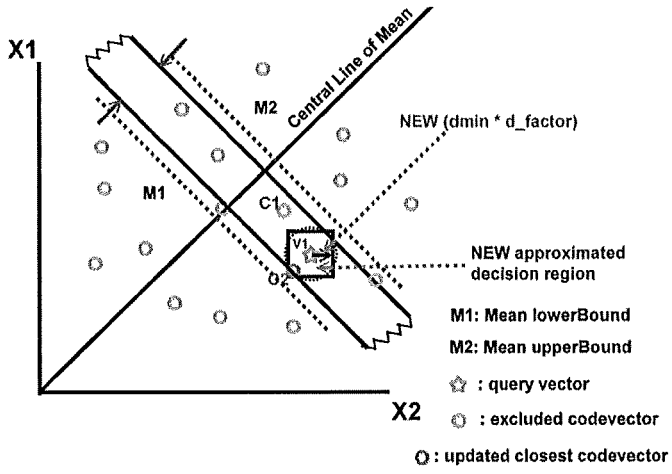


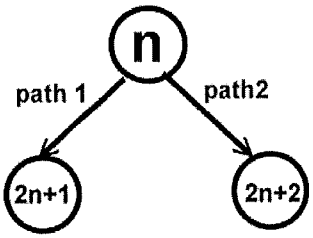
Figure 3.6 The update process of the mean and "hyperbox" boundary

Step 7

When the upper and lower-boundary moved inward, the search area in **figure 3.6** became smaller and more codevectors were not considered. After the remaining testing points inside the search area were evaluated, the process for searching the closest codevector was completed.

3.2 Purpose of some parameters in proposed algorithm

3.2.1 Pre-computed values in Binary Tree Searching



The relationship of the parent node n and its children node is 2n+1 and 2n+2. The query vector *V1* was in position of node n and distance measure was performed to determine which path (i.e.path1 or path2) as the shortest path. However, computing distance is time-consuming and so the pre-computed method was suggested.

If query vector *V1* belongs to node U_{2n+1}, the following equation become true, otherwise it become false and the query vector belongs to node U_{2n+2}

$$\frac{1}{2}\sum_{i=1}^{k}(U_{2n+1_i}-U_{2n+2_i})^2 < \sum_{i=1}^{k}\left[(U_{2n+1_i}-U_{2n+2_i})\times V1_i\right]$$

Where n = 0, 1, 2, 3....M-2

n = node number and M = no. of codevectors

As the number of nodes is fixed and known, $\frac{1}{2}\sum_{i=1}^{k}(U_{2n+1_i}-U_{2n+2_i})^2$ and $(U_{2n+1_i}-U_{2n+2_i})$ can be pre-computed and stored in the codebook. The method of using pre-computed value can save *2*k* multiplication and *2*k + (k-1)* addition for each node where k is dimension of node.

3.2.2 d_factor inside Hyperbox

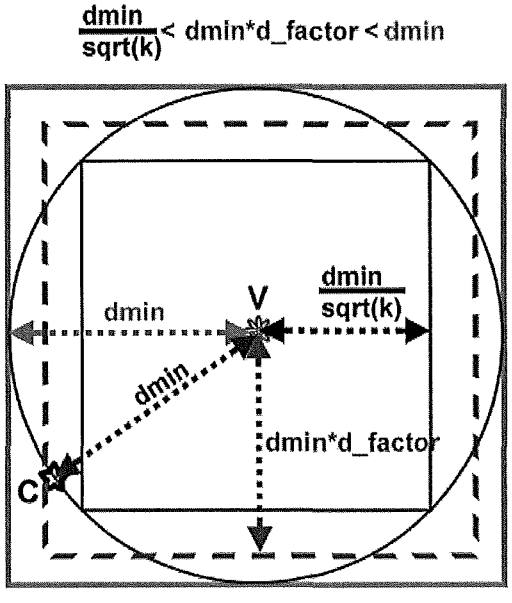


Figure 3.7
d_factor inside the hyperbox

The size of the hyperbox can be controlled by d_factor. The aim of controlling this boundary is to minimize the number of search points provided so that the reconstructed image is nearly the same quality as that by full Search. If the d_factor is set too large, the PSNR will be good but the number of search points will increase. However, if the d_factor is set too small, a fewer number of search points will be tested but the PSNR will unavoidably too low. So, good control of this d_factor is beneficial to balance the number of search points and PSNR

In **figure 3.7**, the range of d_factor can be defined as following equation:

$$\frac{1}{\sqrt{k}} < d\_factor < 1$$ where k is dimension of the vector

The following two sub-sections will discuss about how to set a good d_factor

3.2.2.1 The situation of d_factor under 16 dimension form

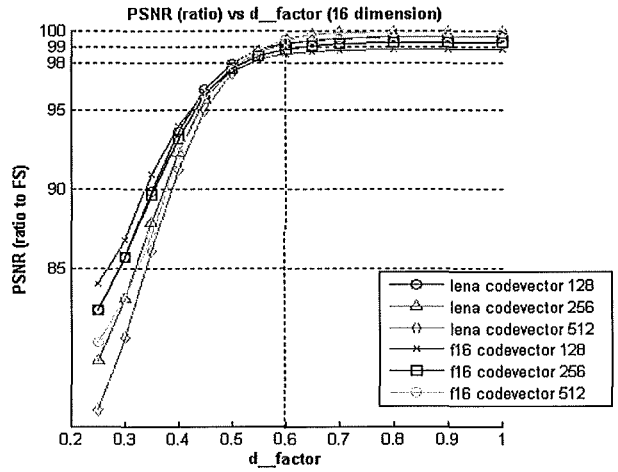


Figure 3.8 PSNR (ratio) VS d_factor (16 dimension)

The image "lena" (inner image) and "f16" (outer image) and the codebook size of 128, 256, 512 are used in this experiment. **Figure 3.8** shows that the curve of PSNR ratio (refer to **Ch. 2.4**) is converged at d_factor of 0.6. If the tolerant PSNR ratio is higher than 0.98, d_factor of 0.6 will be selected for sizing the boundary of hyperbox.

Figure 3.9 and **3.10** illustrate that the bigger the d_factor, the number of search points per query vector will exponentially increase and the number of update points per vector will increase and converge at last.

If the size of hyperbox is restricted by d_factor of 0.6, the average number of search points and update points will be kept below 3 codevectors and 1 codevector respectively. These two values sound attractive as much of calculation was applied on these two types of points. Even by using a bigger size of codebook (i.e. 512), the number of these two types of points is not significantly affected.

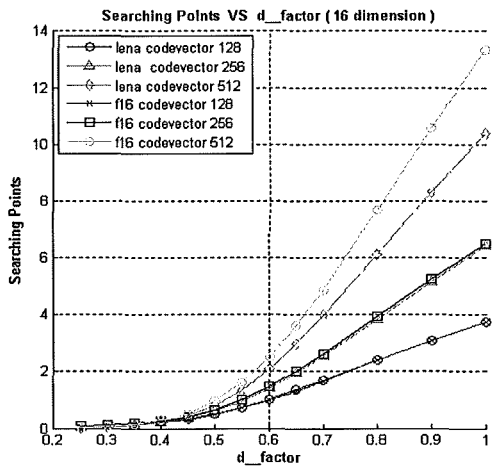


Figure 3.9 Search points VS d_factor (16-dimension)

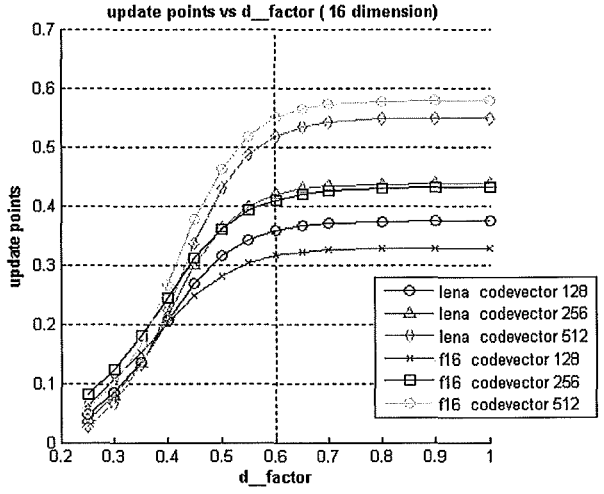


Figure 3.10 Update points VS d_factor (16-dimension)

3.2.2.2 The situation of d_factor under the same codebook size

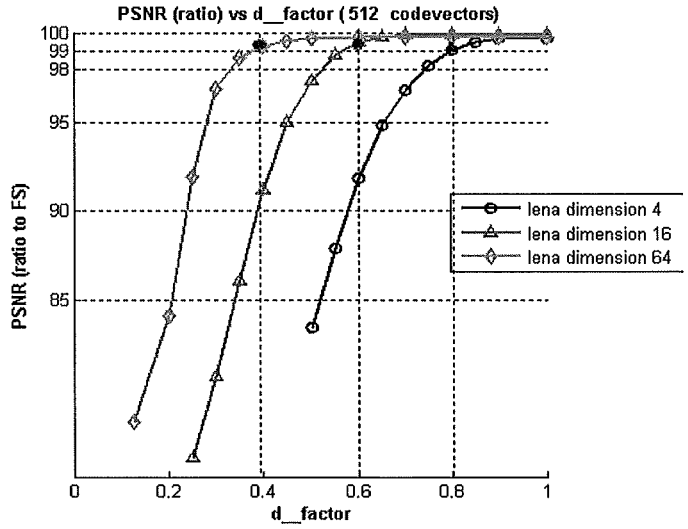


Figure 3.11 PSNR (ratio) VS d_factor (512 codevectors)

The image "lena" (inner image) and the codebook size of 512 codevectors in three different dimensions (4, 16 and 64) were used in this experiment.

Figure 3.11 showed that the three curves of PSNR ratio belonging to 4, 16, 64-dimension converged at different values of d_factor. If the tolerant PSNR ratio is higher than 0.99, then the d_factor for 4, 16, 64-dimension will be 0.8, 0.6 and 0.4 respectively.

Figure 3.12 and **figure 3.13** showed that even under 64-dimension, the number of search points per query vector is around 10 codevectors and that of update points per query vector is below 1 codevector. If a lower dimension is adopted, such as 4-dimension, a very low number of search points (below 1 codevector) can be achieved.

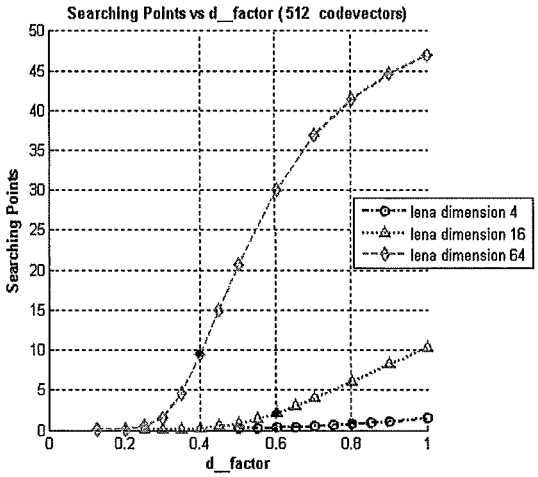


Figure 3.12 Search points VS d_factor (512 codevectors)

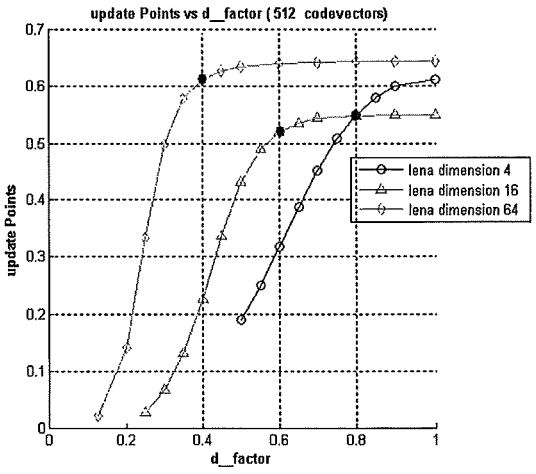


Figure 3.13 Update points VS d_factor (512 codevectors)

3.3 Merits of proposed algorithm

Lower number of search and update points

Our proposed algorithm mainly used Boolean checking rather than using addition and multiplication to exclude far-away codevectors .The following steps summaries how to reduce the number of search and update points.

a) Binary Tree Search

By using a Binary Tree-based search for a query vector in the beginning step, the selected codevector may be the closest codevector or the neighbor of the closest one. It is an essential step because it largely reduces the search area to a great extent as the creation of the initial mean boundary depends on this selected codevector by tree search instead of using the codevector with closest mean. So, the number of the search points is initially small.

b) Pre-compute value in Binary Tree Search

In a binary-tree-based search, to determine which node is the closest one, instead using a partial distance measure, the pre-computed values are used to help lessening the computation complexity.

c) Mean value

Projecting the codevectors on the central line of mean can exclude far-away codevector by examinating the mean value. It is an amazing step as it performs comparison of only one orthogonal value to determine the neighbor codevectors.

d) Hyperbox

Instead of distance measure, "Hyperbox" executes boolean checking of orthogonal values of codevectors and exclude those which orthogonal values are out of boundary. Also, the search area reduced by hyperbox leads to the reduction in further updates. A low number of search and update points implies the low computation complexity.

3.4 Simulation results

To examine the efficiency of the proposed algorithm and, ten images are used to be the candidates of codebook. (see Appendix [1]), we performed some experiments using four **512 x 512** monochrome images: "lena" (inner) , "peppers"(outer), "goldhill" (outer), "f16" (outer).with **256** grey levels. Each image is divided into **4** x **4** blocks, so that the training sequence contains **16384 16**-dimensional vectors.

The proposed algorithm was compared with the **ENNS, MENNS** and **SMENNS** algorithm in terms of search points, update points and speed-up factor

Figure 3.14 shows the strength of our proposed algorithm compared with SMENNS. Tthe number of search points is insensitive to the increase in quantization bits using **4** and **16**-dimensional vectors. Even using the **64**-dimensional vector, the number of search points was relatively low. **Figure 3.15** obviously illustrates the better performance of our proposed algorithm in terms of search points under different number of quantization levels.

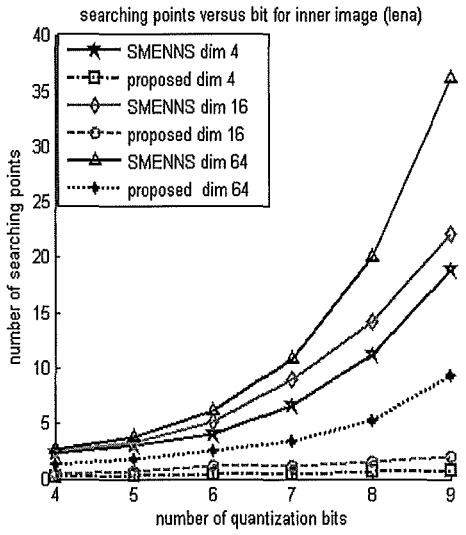


Figure 3.14 search points VS quantization bit under different dimension

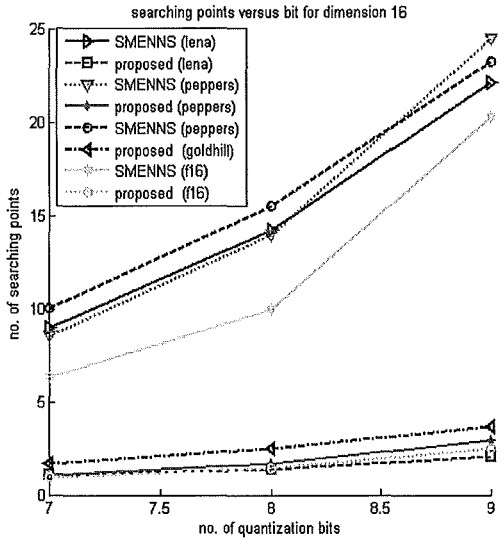


Figure 3.15 search points VS quantization bit using inner and outer images

Table 3.1 shows the speed-up factor (refer to **Ch. 2.2**) of different algorithms under vector form of 16-dimension. Our proposed algorithm can achieve a larger speed-up factor on all three operations when the codebook size increases. It also shows better performance than other three algorithms under different codebook size using inner or outer testing images.

Table 3.1 Comparison of PSNR and Speed up factor for image encoding

Dimension 16	Lena (**inner image**)		Speed up factor		
Codevectors	Method	PSNR	Add	Mul	Bool
128	ENNS	29.8948	7.118373	3.930988	3.424354
-	MENNS	29.8948	6.330639	3.715801	3.214489
-	SMENNS	29.8948	3.617582	6.17602	1.764805
-	**Proposed Algorithm**	**29.8525**	**8.708372**	**6.273105**	**8.216233**
256	ENNS	30.8891	9.673093	5.176352	4.434431
-	MENNS	30.8891	8.628096	5.016006	4.143226
-	SMENNS	30.8891	4.105852	8.012796	1.956481
-	**Proposed Algorithm**	**30.85274**	**15.226**	**10.64193**	**11.03221**
512	ENNS	31.6487	9.758631	5.090745	4.309334
-	MENNS	31.6487	8.777307	5.021296	4.027305
-	SMENNS	31.6487	3.936226	8.291555	1.853309
-	**Proposed Algorithm**	**31.629**	**20.53168**	**14.14454**	**11.45608**

Dimension 16	f16 (**outer image**)		Speed up factor		
Codevectors	Method	PSNR	Add	Mul	Bool
128	ENNS	28.5807	5.854685	3.209809	2.781558
-	MENNS	28.5807	5.196662	3.038727	2.59388
-	SMENNS	28.5807	2.881066	4.962394	1.389665
-	**Proposed Algorithm**	**28.5592**	**7.098469**	**5.225019**	**4.814418**
256	ENNS	29.3282	7.099804	3.758487	3.211856
-	MENNS	29.3282	6.322968	3.648126	2.97408
-	SMENNS	29.3282	3.228333	6.483747	1.52817
-	**Proposed Algorithm**	**29.30895**	**11.868**	**8.626575**	**5.835872**
512	ENNS	29.785	8.238978	4.290397	3.6435
-	MENNS	29.785	7.363473	4.211216	3.369147
-	SMENNS	29.785	3.264395	7.143878	1.535413
-	**Proposed Algorithm**	**29.7659**	**16.34914**	**11.83806**	**6.428741**

The following Reconstructed images showed the performance using our proposed algorithm with 512 codevectors. The pictures on the right side showed the non-black area which is the distortion of image

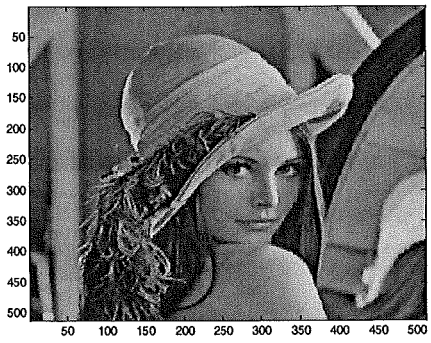


Figure 3.16a

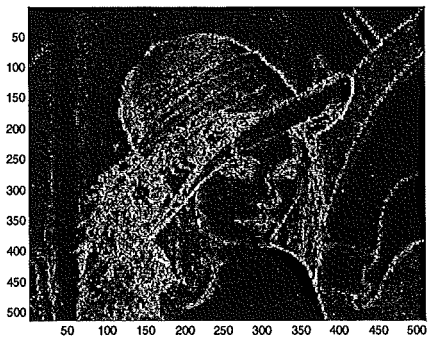


Figure 3.16b

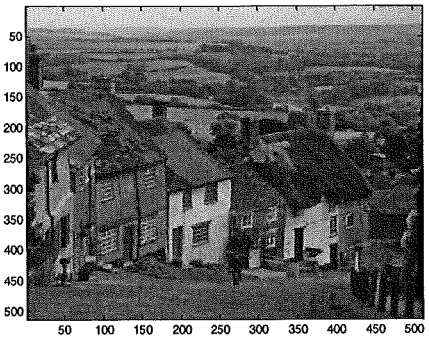


Figure 3.17a

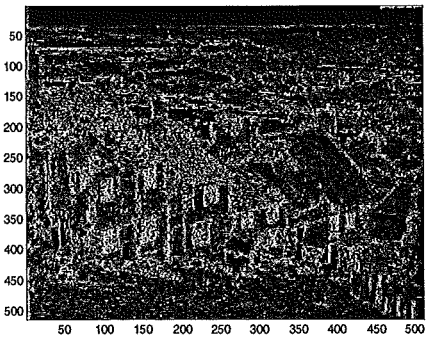


Figure 3.17b

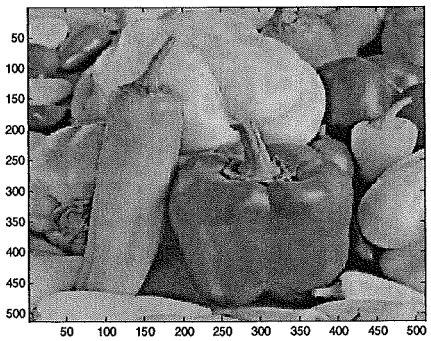


Figure 3.18a

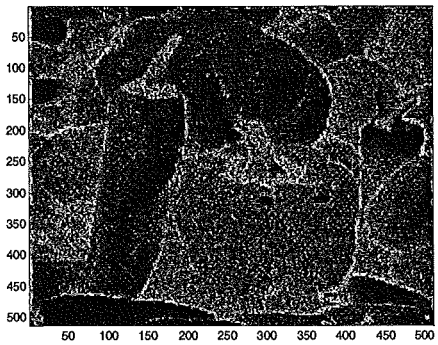


Figure 3.18b

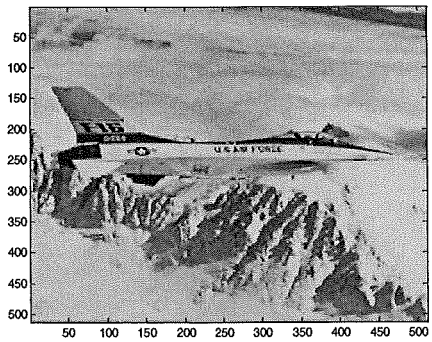


Figure 3.19a

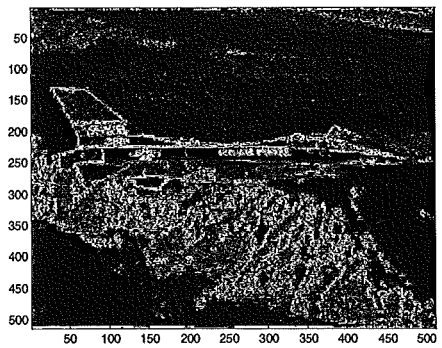


Figure 3.19a

Figure 3.16a (lena), 3.17a (goldhill), 3.18a (peppers), 3.19a (f16) & Figure Nb (distortion figure of Na) where N=3.16,17,18,19

Discussion

The aim of our proposed algorithm was to lessen the computation complexity so as to increase the speed of search for a large set of codevectors. For convenience of evaluation of speed performance, our simulation considered the operators, search points and update points as the candidates of the computation complexity

From the simulation results, it can be concluded that larger set of codevectors have no influence on the computation complexity. The number of codevectors stored in the codebook does not dramatically increase the searching points and update points. The cost of computation on distance evaluation only slightly increase and it is beneficial to a superset of codevectors for searching. For Full Search, if more codevectors are provided for selection, the PSNR will increase and the computation power will increase as well. However, our proposed algorithm is to keep the relative lower cost of computation no matter the increase in the number of codevectors .

Also, the result showed our proposed algorithm performed better on the speed of search. It achieved higher speed up factor compared with other three search algorithms. Even when the number of codevectors increased, the difference between the proposed algorithm and other algorithm tended to be larger. Under the number of 512 codevectors, the PSNR was over 31 dB and the pictures shown were clear and acceptable.

It is suggested that further improvement can be made by removing the DC value of the image before entering the clustering process so that the mean of brightness will be ignored and only pattern of image will be considered. This normalization process helps for extracting more real image patterns and generates a number of more representative codevectors.

References

[1]ORCHARD, M.D.:
'A fast nearest-neighbour search algorithm',
IEEE ICASSP

[2] LEE, C.-H., and CHEN, L.-H.:
'Fast closest codeword search algorithm for vector quantization',
IEE Proc. Vis. Image SignalProcess., 1994

[3] D.Ghosh & A. P. S h iva prasad
'Fast codeword search algorithm for real-time codebook generation in adaptive VQ'
Vision, Image and Signal Processing, IEE Proceedings Oct 1997

[4] GHOSH, D., and SHIVAPRASAD, A.P.:
'Fast equal average nearest neighbour codeword search algorithm for vector quantization'.
Proceedings of the National Conference on Communication, 1996, Bombay, India,

[5] GRAY, R.M.:
'Vector quantization',
IEEE ASSP Mag., 1984

[6] POGGI, G.:
'Fast algorithm for full-search VQ encoding',
Electron.Lett., 1993, **29**

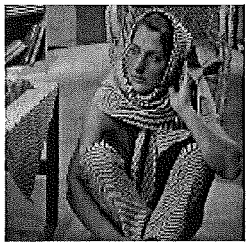
Appendix

[1] ten candidates of image: lena. ,tiffany, elaine, woman, barbara, girl, zelda, couple, man, crowd

man

crowd

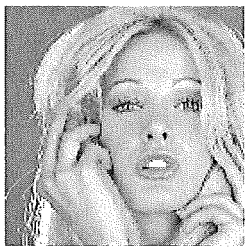
barbara



elaine

lena

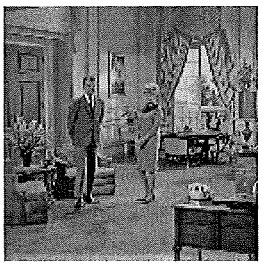
tiffany



woman

zelda

couple



girl