Performance Improvement of Vector Quantization with Bit-parallelism Hardware

Pi-Chung Wang pcwang@cs.nchu.edu.tw
Institute of Networking and Multimedia
and Department of Computer Science and Engineering
National Chung Hsing University
Taichung, 402, Taiwan, R.O.C.

Abstract

Vector quantization is an elementary technique for image compression; however, searching for the nearest codeword in a codebook is time-consuming. In this work, we propose a hardware-based scheme by adopting bit-parallelism to prune unnecessary codewords. The new scheme uses a "Bit-mapped Look-up Table" to represent the positional information of the codewords. The lookup procedure can simply refer to the bitmaps to find the candidate codewords. Our simulation results further confirm the effectiveness of the proposed scheme.

Keywords: Image Compression, Nearest Neighbor Search, Vector Quantization, Look-up Tables.

1. INTRODUCTION

The use of images has become a common practice in computer communications. The sizes of images are usually huge and, therefore, need to be compressed for storage and transmission efficiency. Vector quantization (VQ) [1] is an important technique for image compression and has been proven to be simple and efficient [2]. VQ can be defined as a mapping from k-dimensional Euclidean space into a finite subset C. The finite set C is known as the *codebook* and $C=\{c_i|i=1,2,\ldots,N\}$, where c_i is a codeword and N is the codebook size.

To compress an image, VQ comprises two functions: an encoder and a decoder. The VQ encoder first divides the image into $N_w \times N_h$ blocks (or vectors). Let the block size be k ($k=w\times h$), and then each block is a k-dimensional vector. VQ selects a codeword $c_q=[c_{q(0)},c_{q(1)},\ldots,c_{q(k-1)}]$ for each image vector $x=[x_{(0)},x_{(1)},\ldots,x_{(k-1)}]$ such that the distance between x and c_q is the smallest, where c_q is the closest codeword of x and $c_{q(j)}$ denotes the j_{th}-dimensional value of the codeword c_q. The distortion between the image vector x and each codeword c_i is measured by their *squared Euclidean distance*, i.e.,

$$d(x,c_i)=\|x-c_i\|^2=\sum_{j=0}^{k-1}\left[x_{(j)}-c_{i(j)}\right]^2 \tag{1}$$

After the selection of the closest codeword, VQ replaces the vector x by the index q of c_q. The VQ decoder has the same codebook as that of the encoder. For each index, VQ decoder can easily fetch its corresponding codeword, and piece them together into the decoded image.

The codebook search is one of the major bottlenecks in VQ. From Equation (1), the calculation of the squared Euclidean distance needs k subtractions and k multiplications to derive $k[x_{(j)}-c_{i(j)}]^2$. Since the multiplication is a complex operation, it increases the total computational complexity of Equation (1). Therefore, speeding up the calculation of the squared Euclidean distance is a major hurdle.

Owing to the importance of vector quantization, a handful of methods have been proposed to shorten VQ encoding time [3,5–8,14,16,17]. The simplest one among them is the *look-up table* (LUT) method [6,17]. It suggests that the results of $[x_{(j)}-c_{i(j)}]^2$ for all possible x_j and y_{ij} should be pre-computed first and then stored into a huge matrix, the *LUT*. Suppose the values of $x_{(j)}$ and $c_{i(j)}$ are within $[0,m-1]$. Then the size of matrix *LUT* should be $m\times m$ and

$$LUT=\begin{bmatrix} 0 & 1^2 & \cdots & (m-1)^2 \\ 1^2 & 0 & \cdots & (m-2)^2 \\ \vdots & \vdots & \ddots & \vdots \\ (m-1)^2 & (m-2)^2 & \cdots & 0 \end{bmatrix}_{m\times m} \tag{2}$$

Given any $x_{(j)}$ and $c_{i(j)}$, we can get the square of their difference directly from $LUT[x_{(j)}, c_{i(j)}]$. Therefore, Equation (1) could be rewritten as follows:

$$d(x,c_i)=\sum_{j=0}^{k-1}\left[x_{(j)}-c_{i(j)}\right]^2=\sum_{j=0}^{k-1}LUT\left[x_{(j)},c_{i(j)}\right] \tag{3}$$

LUT can be employed to avoid the subtraction and the multiplication in Equation (1). Hence, it is an efficient method.

Rizvi et. al. proposed another LUT-based scheme in [8] to fasten the calculation of squared Euclidean distances which is called *truncated look-up table* (TLUT). In their method, Rizvi et. al. pre-computed the results of $(x_{(j)}-c_{i(j)})^2$ for all possible $|x_{(j)}-c_{i(j)}|$ and stored these results into the matrix *TLUT*, where

$$TLUT=\left[0,1^2,2^2,\ldots,(m-1)^2\right]_{m\times 1} \tag{4}$$

Since the calculation complexity of the absolute subtraction $|x_{(j)}-c_{i(j)}|$ operation is much simpler and more efficient than multiplication operation, it could be derived before accessing the LUT. Hence, according to the matrix *TLUT*, Equation (1) can be expressed as follows:

$$d(x,c_i)=\sum_{j=0}^{k-1}\left[x_{(j)}-c_{i(j)}\right]^2=\sum_{j=0}^{k-1}TLUT\left\|x_{(j)}-c_{i(j)}\right\| \tag{5}$$

The size of the matrix *TLUT* is *m*. Yet, it is still difficult for some special designs, such as VLSI implementations and systolic architectures [5,7], to be implemented.

In short, the design criteria of LUT-based schemes emphasize computation speed, table storage and image quality. However, the number of the calculated codewords has not been investigated since these schemes did not use the geometrical information implied in the codewords.

In this work, we propose a hardware-based scheme to represent the positional information. This scheme adopts bit-parallelism and constructs a “Bit-mapped Look-up Table” to prune feasible codewords. The lookup procedure simply involves the use of the bitmap information to prune candidate codewords. Moreover, the scheme is a plug-in, which can cooperate with other existing schemes to further tune up the search speed. An efficient hardware implementation is also presented in our simulation results.

The rest of this paper is organized as follows. The proposed scheme is presented in Section 2. Section 3 addresses the performance evaluation. Section 4 concludes the work.

2. BIT-PARALLEISM SCHEME

As discussed in previous section, VQ matches a test vector to the codeword with the smallest Euclidean distance. The selected codeword could be treated as the nearest point in the *k*-dimensional space. In the ideal case (with a well-trained codebook), the distance of each dimension between the test vector and the selected codeword should be very close. Hence, it is possible to filter out unfeasible codewords by referring to the positional information.

We use a codebook and five different images to show the effect of one-dimensional distances on andthe quality of the compressed images. The codebook is trained by using an image of "Lena" and five other images which are quantized by full search. The image quality is measured by *peak signal-to-noise ratio* (PSNR), which is defined as

$$PSNR = 10 \times \log_{10}(255^2 / MSE) \text{ dB}. \tag{6}$$

The *mean-square error* (MSE) is defined as

$$MSE = (1/H) \times (1/W) \sum_{i=0}^{H-1} \sum_{j=0}^{W-1} \left[\alpha_{(i,j)} - \beta_{(i,j)} \right]^2 \tag{7}$$

for an $H \times W$ image, where $\alpha_{(i,j)}$ and $\beta_{(i,j)}$ denote the original and quantized gray levels of pixel (i,j) in the image, respectively. A larger PSNR value usually preserves the original image quality better.

We shown the distribution of the maximal one-dimensional distance, $max|x_{(j)}-c_{M(i)}|$, where $0 \leq j \leq k-1$, between the test vectors and their matched codewords for each image in Fig. 1. For high-quality compressed images "Lena" and "Zelda", about 98% of their maximum one-dimensional distances are less than 32. However, the ratio is reduced to 91%~96% for other compressed images with lower quality.

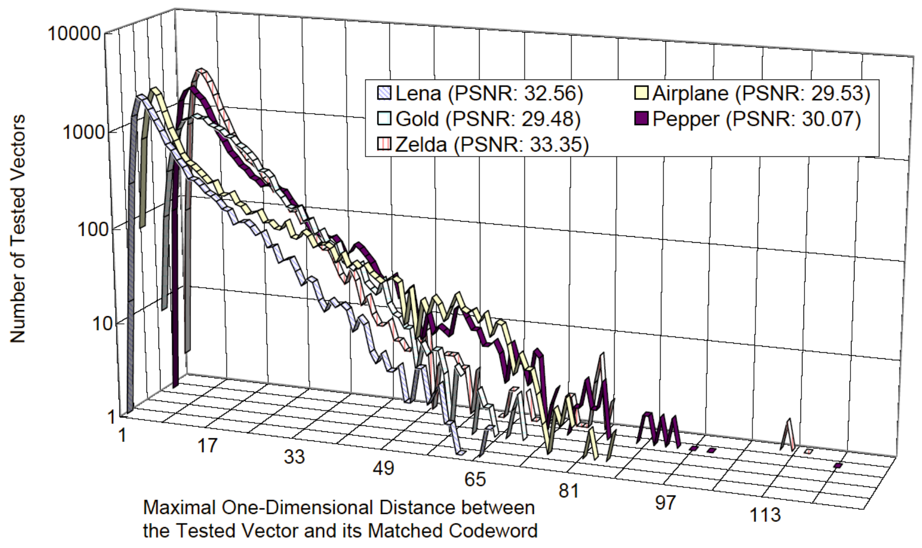

FIGURE 1: Distribution of the Maximum One-dimensional Distances for Different Images.

Figure 1 demonstrates that it is possible to prune feasible codewords by using only one-dimensional distances, especially for those images with high compression quality. We further demonstrate how one-dimensional distances can improve the performance of VQ with an example in Fig. 2. There are two codewords, C_1 (3, 1) and C_2 (2, 3) in this example. To calculate the nearest codeword for the test vector, V_1 (1, 2), the squared Euclidean distances to C_1 and C_2 are 5 and 2, respectively, and C_2 is chosen as the result. If we only consider the horizontal distances, only C_2 would be selected without performing the calculation for the Euclidean distance. However, both codewords are chosen by taking only the vertical distances into consideration, and the Euclidean distances to these two vectors must be calculated in the same way as in the full search to decide the closest codeword. In addition, using one dimension to select the candidate codewords might cause false matches. Let's consider a two-dimensional example in Fig. 2. In Fig. 2, C_2 is closer to V_2 than C_1 in the first dimension. Yet, the Euclidean distance between C_1 and V_2 is smaller than that between C_2 and V_2. Comparing multiple codewords, which are within a certain range in the dimension, could alleviate this problem.

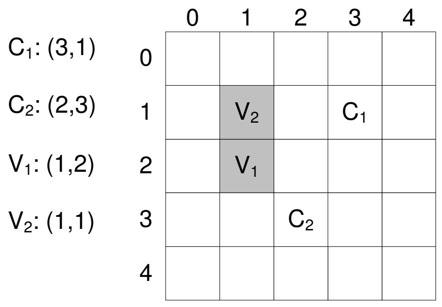

FIGURE 2: A Two-dimensional Example.

In sum, when the codeword is positioned closer to the test vector in the Euclidean space, the distance in each dimension is likely to be shortened as well. To exploit this property, we adopt bitmaps to represent the positional information. For each codeword, there is a uni-codeword bitmap with m bits for each selected dimension *j*, where $0 \leq j \leq k-1$. Each bit in the uni-codeword bitmap corresponds to a position in dimension *j*. Assume that the pre-defined distance is *D*. The bits from $c_{i(j)-D}$ to $c_{i(j)+D}$ are set to one for codeword *i*.

Figure 3 shows the resulting bitmaps for the example in Fig. 2. The distance *D* is defined as 1 for both dimensions. The set bits form a square in the two-dimensional case. If the test vector is located within the square of a certain codeword, the codeword would be one of the candidates in vector quantization. For example, V_1 is within the square of C_2, rather than C_1. Thus C_1 would not be considered in vector quantization. The bricks, (2,2) and (3,2), are shared by the squares of C_1 and C_2. Therefore, the test vectors positing in these two bricks select C_1 and C_2 as the candidates.

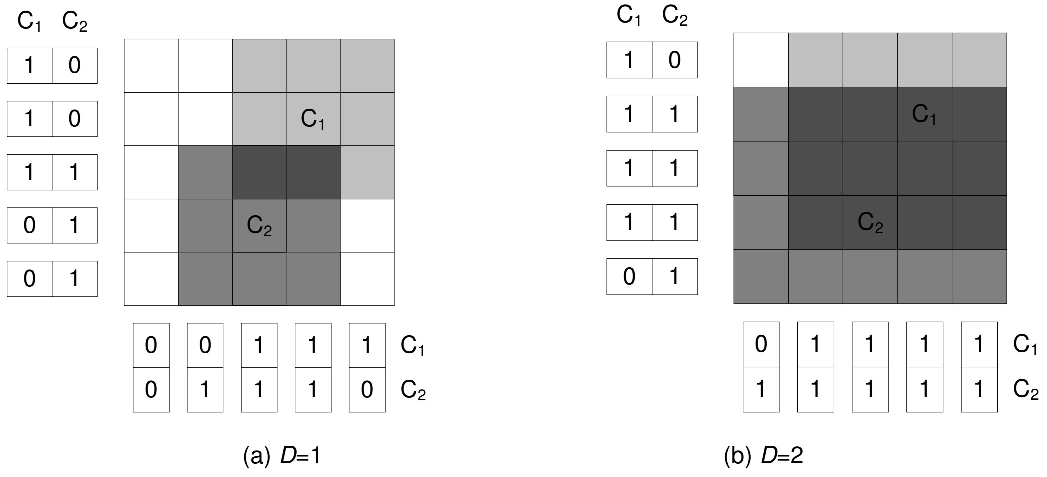

FIGURE 3: Two-dimensional Uni-codeword Bitmaps.

As seen in Fig. 3(a), there are several unoccupied bricks remaining. For the test vectors located within these bricks, e.g. V_2, the bitmaps are useless since there is no candidate could be derived. As a result, each codeword has to be calculated to nail down the one with the smallest Euclidean distance. To ease the problem, a longer distance could be adopted, as shown in Fig. 3(b), where the renewed bitmaps for *D*=2 are presented. With the new distance, most bricks are occupied by at least one codeword's square. However, the conjunct bricks are also increased due to the enlarged squares. A suitable distance is thus important to the performance of the proposed scheme since a wider range could increase the number of candidates, whereas a shorter distance might result in a null set. In our experiments, various distances are investigated to evaluate the performance and the image quality. Next, the construction/lookup procedure of the searchable data structure is introduced.

2.1 The Construction of the Searchable Data Structure - Positional Bitmaps

Although our uni-codeword bitmaps could present the positional information of each codeword, they are not searchable. It is because accessing uni-codeword bitmaps for each codeword would be inefficient. To facilitate the lookup speed, the uni-codeword bitmaps are transformed to uni-position bitmaps, which record the one-dimensional positions of related codewords. The relationships between the uni-codeword bitmaps and the uni-position bitmaps are illustrated in Fig. 4. The uni-position bitmap for position *p* at dimension *j* is defined as $B^D_{j,p}$, where *D* is the preset distance. The i_{th} bit is defined as $B^D_{j,p}(i)$ which is set to one if $p\text{-}D \le c_{i(j)} \le p+D$. The pseudo code is given in Fig. 5. For each bitmap, the required storage is $m \times N$ per dimension. Since there are *k* dimensions, each bitmap requires $m \times N \times k$ bits. Therefore, each bitmap consumes 8 kilobytes for a typical 256-codeword codebook with 256 gray levels.

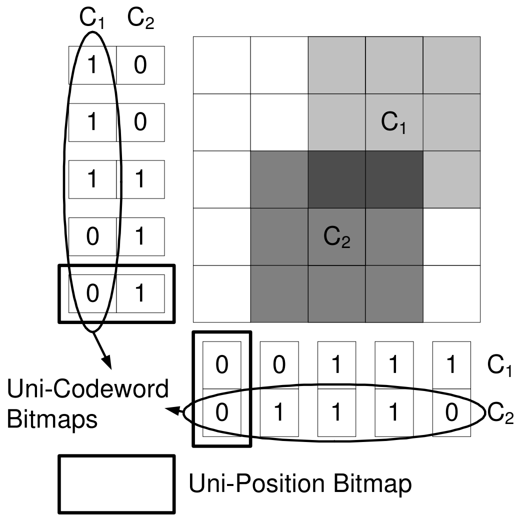

FIGURE 4: Relationships Between Uni-codeword Bitmaps and Uni-position Bitmaps. (*D*=1)

```
Bitmap-Filling Algorithm
For each uni-position bitmap $B^D_{j,p}$ BEGIN
  For each bit $B^D_{j,p}(i)$ BEGIN
    If $p-D \le c_{i(j)} \le p+D$, $B^D_{j,p}(i)=1$.
    Otherwise, $B^D_{j,p}(i)=0$.
  END
END
```

FIGURE 5: The Bitmap-Filling Algorithm

2.2 The Lookup Procedure

Our scheme, bit-mapped look-up table (BLUT), combines bitmap pruning and TLUT to achieve fast processing. For a test vector in the BLUT scheme, the j_{th} value x_j is used to access the bitmap B^D_{j,x_j}. Each set bit indicates that the corresponding codeword is within a distance *D* from the test vector at dimension *j*. Accordingly, the Euclidean distance is calculated by accessing TLUT. The pseudo code for the lookup procedure is listed in Fig. 6. To check whether the i_{th} bit is set, the fetched bitmap B^D_{j,x_j} is intersected to a pre-generated bitmap with only i_{th} bit set (00…010…0) by performing "**AND**" operation. If the value is larger than zero, then the codeword *i* is the candidate.

```
Vector Quantization by BLUT Algorithm
For each vector x BEGIN
  Fetch the $B^D_{j,x_j}$.
  For each set bit $B^D_{j,x_j}(i)$ BEGIN
    Calculate Euclidean distance $d(x,c_i)$ where
    $d(x,c_i)=\sum_{j=0}^{k-1} TLUT\left\|x_{(j)},c_{i(j)}\right\|$.
    If $d(x,c_i) \le$ min_distance BEGIN
      min_distance_id= i
      min_dietance= $d(x,c_i)$
    END
  END
  min_distance_id is the quantized index for x
END
```

FIGURE 6: Vector Quantization by BLUT Algorithm

We use the previous example in Fig. 4 with D=1 to explain the lookup procedure. Assume that the horizontal dimension is used to prune the codebook. For the test vector V_1 (1,2), the second uni-position bitmap (0,1) is fetched. Therefore, C_1 is not considered in the vector quantization and C_2 is the result of vector quantization. However, for another vector V_2 (1,1), a *false match* is caused since only C_2 is selected as the candidate but C_1 is closer to V_2 than C_2. This is mainly due to bitmaps of only one dimension sometimes cover insufficient information.

The first variation is to generate bitmaps for multiple dimensions. With more bitmaps, the effect on codebook-pruning is increased. Furthermore, the multi-dimensional BLUT algorithm can improve the accuracy of image compression since it can cover more information as compared with the one-dimensional scheme. However, the required storage is also increased proportionately. The lookup procedure for multiple bitmaps is shown in Fig. 7. The set *dim* records the selected dimensions where $1 \le |dim| \le k$. Before the set bits are checked, the multiple bitmaps are intersected by performing "**AND**" operation to derive the representative bitmap R^D.

Again, we use the previous example to explain the procedure. For the test vector V_1 (1,2), the second uni-position bitmap (0,1) at the horizontal axis and third one (1,1) at the vertical axis are fetched. Consequently, the bitmap (0,1) is derived by intersecting two bitmaps, and C_2 is selected as the result. To quantize V_2, the resulting bitmap is (0,0), which means no candidates have been found. As a result, the Euclidean distance to C_1 and C_2 are calculated to decide the closest codeword. Although an extra step is required in the computation, *false matches* are eliminated.

```
Multi-dimensional BLUT Algorithm
For each vector x BEGIN
  Fetch the $B^D_{j,x_j}$, where $j \in dim$
  $R^D=\cap_{j\in dim} B^D_{j,x_j}$
  For each set bit $R^D(i)$ BEGIN
    Calculate Euclidean distance $d(x,c_i)$ where
    $d(x,c_i)=\sum_{j=0}^{k-1} TLUT\left\|x_{(j)},c_{i(j)}\right\|$.
    If $d(x,c_i) \le$ min_distance BEGIN
      min_distance_id= i
      min_dietance= $d(x,c_i)$
    END
  END
  min_distance_id is the quantized index for x
END
```

FIGURE 7: Multi-dimensional BLUT Algorithm

BLUT could also adopt multiple distances to derive a minimum set of candidates. The basic operation is to test the bitmaps with D=1. If there is no candidate, then the bitmaps for D=2 is tested and so forth. In the basic scheme, the time complexity for bitmap operation is O(m). Simple binary search on distance could further improve the performance. That is, the bitmaps for $D=m/2$ are tested at first. If the candidate set is empty, then the bitmaps for $D=m/4$ are tested. Otherwise, the $D=(m-1+m/2)/2$ bitmaps are examined. The time complexity is thus reduced to O($\log_2 m$). However, such scheme requires huge memory space, thus further reduction of storage is necessary.

2.3 Implementations

In the one-dimensional version, BLUT is suitable for software implementation due to its simplicity. Nevertheless, hardware implementation is preferable for multi-dimensional BLUT since it requires memory bus with N-bit wide (typically N=256). In Fig. 8, we present a conceptual model for hardware implementation. This implementation includes independent RAM modules for uni-position bitmaps. Bitmaps of a specific dimension are located in a storage. While performing lookup, the uni-position bitmaps are fetched from RAM modules simultaneously and perform the "AND" operation. Accordingly, the resulting bitmap R^D enables the codewords in the candidate set for calculating the Euclidean distance in ALU.

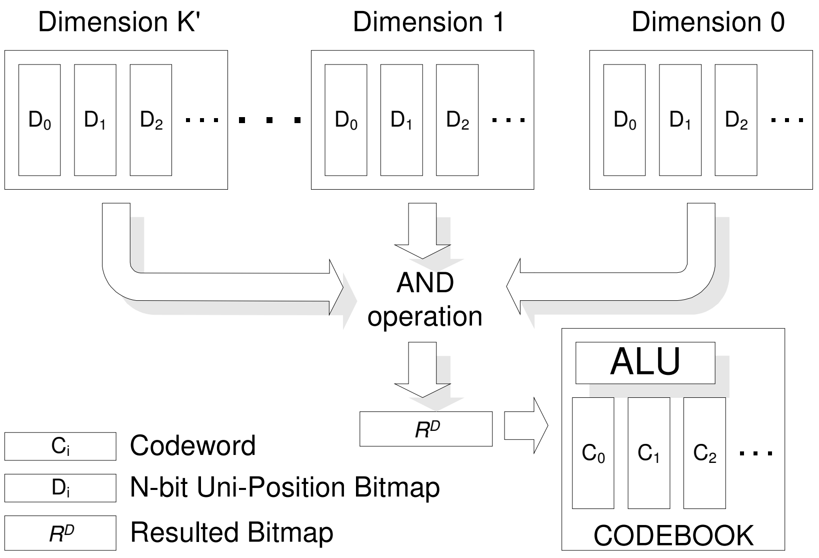

FIGURE 8: Hardware Implementation for Multi-dimensional BLUT.

3. SIMULATION RESULTS

We have conducted several simulations to show the efficiency of BLUT. All images used in these experiments were 512×512 monochrome still images, with each pixel of these images containing 256 gray levels. These images were then divided into 4×4 pixel blocks. Each block was a 16-dimensional vector. We used image "Lena" as our training set and applied the Lindo-Buzo-Gray (LBG) algorithm to generate our codebook C. In the previous literature [1,3], the quality of an image compression method was usually estimated by the following five criteria: compression ratio, image quality, execution time, extra memory size, and the number of mathematical operations. All of our experimental images had the same compression ratio. Thus only the latter four criteria are employed to evaluate the performance. The quality of the images is estimated by PSNR. The extra memory denotes the storage needed to record the projected space and the projected values from codewords onto the projected space. As for the mathematical operations, the number of calculated codewords is considered since the operations for each codeword are identical.

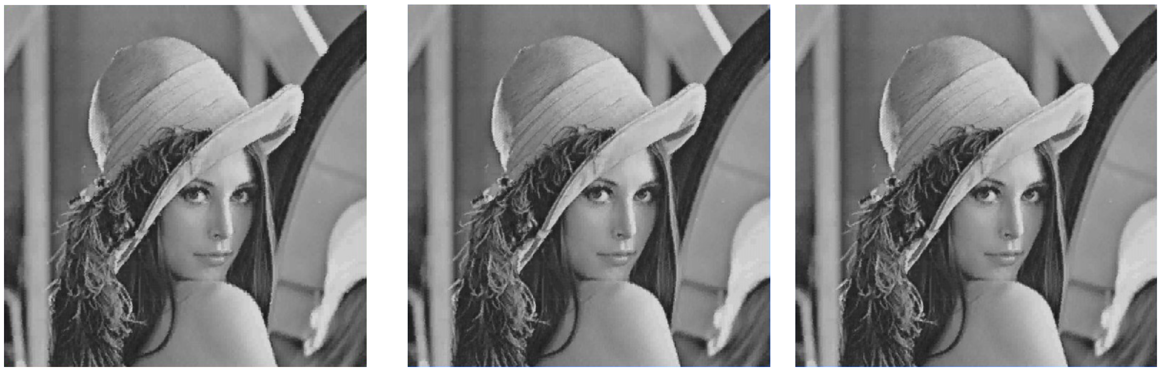
(a) *D*=16 (PSNR=31.329) (b) *D*=32 (PSNR=32.374) (c) *D*=64 (PSNR=32.554)

FIGURE 9: The Lena Images of BLUT.

In the first part, the performance of the software-based implementation with one bitmap was investigated. The experiments were performed on an IBM PC with a 500-MHz Pentium CPU. Table 1 shows the experimental results of BLUT based on image “Lena”. VQ indicates the vector quantization without any speedup. The distances for BLUT vary from 16 to 128. With a shorter distance, the image quality is degraded since the occurrence of false matches is increased. Nevertheless, the calculated codewords are reduced by using uni-position bitmap as well as the execution time. Full search requires no extra storage while TLUT needs 256 bytes. For BLUT, the extra storage of a bitmap is 8 kilobytes and 256 bytes for TLUT. The decompressed images are shown in Fig. 9. While BLUT’s distance is extended to 64, the image quality is almost identical to VQ and TLUT.

Metrics	Full Search	TLUT	BLUT Scheme				
			D=16	D=32	D=48	D=64	D=128
PSNR	32.56	32.56	31.329	32.374	32.53	32.554	32.56
Execution Time (sec.)	1.72	1.65	0.40	0.77	0.99	1.31	1.84
Calculated Codewords	256	256	50	93	131	165	243
Memory Size (Bytes)	0	256	8192 (Uni-position Bitmap) + 256 (TLUT)				

TABLE 1: The Performance of the Software-based BLUT. (Image: Lena)

Next, we adopt multiple bitmaps to evaluate the performance of the proposed BLUT schemes. Since software platform cannot support wide memory bus, the proposed scheme with multiple bitmaps is based on hardware implementation. The bitwise operation could be parallelized and the time for calculating Euclidean distance is proportional to the number of calculated codewords. In our experiments, we examine the performance with 1, 2 and 4 bitmaps. The distance varies from 16 to 128. In addition, we also present the performance of BLUT with different codebook sizes (128, 256, 512 and 1024) in Tables 2~5.

Table 2 lists the numerical results of vector quantization with 128 codewords. With distance D equals 16, the PSNR value degrades severely as the number of bitmaps increases to 4. This is because a smaller codebook would increase the resulted Euclidean distance of vector quantization as well as the maximal one-dimensional distance. Hence, the effect of BLUT is lessened since greater distance is required. If the distance extends to 32, the resulted image quality by using more than one bitmap is significantly improved. Moreover, only about one fifth of the codewords is calculated for Euclidean distance. BLUT with two bitmaps is usually superior to that with four bitmaps. However, as the number of codewords increases, the 4-bitmap BLUT could outperform the 2-bitmap BLUT by calculating fewer codewords while retaining similar image quality, as demonstrated in Table 5.

Images		Airplane			Gold			Lena			Pepper			Zelda		
Metrics		Code-words	Time	PSNR	Code-words	Time	PSNR	Code-words	Time	PSNR	Code-words	Time	PSNR	Code-words	Time	PSNR
FS		128	0.54	28.97	128	0.54	28.54	128	0.53	31.46	128	0.55	28.89	128	0.54	32.31
TLUT		128	0.43	28.97	128	0.42	28.54	128	0.42	31.46	128	0.42	28.89	128	0.42	32.31
16	1	23	0.08	28.15	20	0.07	28.12	23	0.08	30.47	21	0.07	24.69	23	0.08	31.89
	2	7	0.02	26.02	5	0.02	27.09	7	0.02	28.30	7	0.02	19.58	7	0.02	29.85
	4	6	0.02	23.12	4	0.01	24.60	6	0.02	24.62	5	0.02	18.62	6	0.02	26.53
32	1	43	0.14	28.78	39	0.13	28.50	43	0.14	31.32	42	0.14	28.49	44	0.14	32.22
	2	22	0.07	28.36	19	0.06	28.45	22	0.07	30.91	22	0.07	27.78	23	0.08	31.91
	4	19	0.06	27.09	16	0.05	28.08	19	0.06	29.81	19	0.06	27.03	20	0.07	31.56
64	1	74	0.24	28.96	74	0.25	28.54	80	0.27	31.46	76	0.25	28.87	80	0.26	32.31
	2	51	0.17	28.96	55	0.18	28.54	59	0.20	31.45	56	0.18	28.85	61	0.20	32.30
	4	49	0.16	28.95	51	0.17	28.54	56	0.19	31.45	54	0.18	28.84	58	0.19	32.28
128	1	114	0.37	28.97	118	0.40	28.54	120	0.40	31.46	117	0.38	28.89	121	0.40	32.31
	2	102	0.33	28.97	112	0.38	28.54	114	0.38	31.46	111	0.36	28.89	117	0.38	32.31
	4	101	0.33	28.97	110	0.37	28.54	113	0.38	31.46	110	0.36	28.89	116	0.38	32.31

TABLE 2: The Performance of the Hardware-based BLUT. (*N*=128)

Images		Airplane			Gold			Lena			Pepper			Zelda		
Metrics		Code-words	Time	PSNR	Code-words	Time	PSNR	Code-words	Time	PSNR	Code-words	Time	PSNR	Code-words	Time	PSNR
FS		256	1.30	29.53	256	1.30	29.48	256	1.30	32.56	256	1.29	30.07	256	1.30	33.35
TLUT		256	1.11	29.53	256	1.10	29.48	256	1.09	32.56	256	1.10	30.07	256	1.09	33.35
16	1	36	0.15	28.62	46	0.20	29.12	50	0.22	31.33	42	0.18	25.95	51	0.22	32.98
	2	11	0.05	26.86	15	0.07	28.27	19	0.08	29.13	15	0.06	21.09	20	0.09	31.39
	4	9	0.04	23.81	10	0.04	25.64	15	0.07	25.66	12	0.05	20.07	16	0.07	29.46
32	1	66	0.28	29.30	88	0.38	29.45	93	0.40	32.37	84	0.36	29.73	97	0.42	33.27
	2	33	0.14	28.93	48	0.21	29.40	54	0.23	32.08	48	0.20	29.12	59	0.25	33.03
	4	29	0.12	28.03	40	0.17	29.21	47	0.20	31.04	42	0.18	28.16	52	0.22	32.59
64	1	134	0.57	29.53	161	0.70	29.48	165	0.72	32.55	160	0.68	30.04	172	0.74	33.34
	2	98	0.42	29.52	128	0.56	29.48	132	0.57	32.55	127	0.54	30.01	141	0.61	33.30
	4	92	0.39	29.52	119	0.52	29.48	125	0.54	32.55	121	0.52	30.00	135	0.58	33.29
128	1	224	0.95	29.53	240	1.04	29.48	243	1.05	32.56	239	1.02	30.07	247	1.06	33.35
	2	207	0.88	29.53	232	1.01	29.48	236	1.02	32.56	230	0.98	30.07	242	1.04	33.35
	4	206	0.88	29.53	230	1.00	29.48	234	1.02	32.56	227	0.97	30.07	240	1.03	33.35

TABLE 3: The Performance of the Hardware-based BLUT. (*N*=256)

Images		Airplane			Gold			Lena			Pepper			Zelda		
Metrics		Code-words	Time	PSNR	Code-words	Time	PSNR	Code-words	Time	PSNR	Code-words	Time	PSNR	Code-words	Time	PSNR
FS		512	2.39	30.00	512	2.41	30.18	512	2.39	33.63	512	2.42	30.57	512	2.41	34.08
TLUT		512	2.15	30.00	512	2.16	30.18	512	2.16	33.63	512	2.14	30.57	512	2.14	34.08
16	1	66	0.28	29.33	90	0.38	29.85	99	0.42	32.69	84	0.35	26.12	103	0.44	33.75
	2	25	0.11	27.70	34	0.14	29.22	42	0.18	30.61	34	0.14	21.53	46	0.19	32.19
	4	19	0.08	24.61	23	0.10	26.77	33	0.14	26.98	27	0.11	20.50	37	0.16	30.46
32	1	126	0.53	29.85	175	0.74	30.15	182	0.77	33.45	167	0.70	30.32	193	0.82	34.05
	2	70	0.30	29.45	103	0.43	30.12	113	0.48	33.03	101	0.42	29.87	124	0.52	33.73
	4	60	0.25	28.43	86	0.36	29.95	100	0.42	31.63	89	0.37	29.12	112	0.47	33.09
64	1	261	1.10	30.00	322	1.35	30.18	326	1.37	33.62	316	1.32	30.56	340	1.44	34.08
	2	197	0.83	29.99	262	1.10	30.18	266	1.12	33.62	258	1.08	30.51	285	1.20	34.08
	4	185	0.78	29.99	245	1.03	30.18	253	1.06	33.62	247	1.03	30.51	274	1.16	34.07
128	1	437	1.85	30.00	479	2.01	30.18	485	2.04	33.63	475	1.99	30.57	495	2.09	34.08
	2	407	1.72	30.00	465	1.96	30.18	472	1.98	33.63	458	1.92	30.57	486	2.05	34.08
	4	402	1.70	30.00	461	1.94	30.18	468	1.97	33.63	454	1.90	30.57	482	2.04	34.08

TABLE 4: The Performance of the Hardware-based BLUT. (*N*=512)

Images		Airplane			Gold			Lena			Pepper			Zelda		
Metrics		Code-words	Time	PSNR	Code-words	Time	PSNR	Code-words	Time	PSNR	Code-words	Time	PSNR	Code-words	Time	PSNR
FS		1,024	4.93	30.43	1,024	4.93	30.80	1,024	4.87	34.42	1,024	4.93	30.83	1,024	4.90	34.51
TLUT		1,024	4.39	30.43	1,024	4.39	30.80	1,024	4.40	34.42	1,024	4.39	30.83	1,024	4.40	34.51
16	1	146	0.63	29.79	178	0.76	30.55	199	0.85	33.59	171	0.73	26.63	205	0.88	34.26
	2	66	0.28	28.39	74	0.32	29.96	98	0.42	31.64	79	0.34	21.70	103	0.44	32.71
	4	51	0.22	25.28	50	0.21	27.56	78	0.33	27.80	62	0.27	20.74	83	0.36	31.17
32	1	269	1.15	30.30	346	1.49	30.78	369	1.58	34.33	337	1.45	30.62	386	1.65	34.47
	2	164	0.70	30.01	218	0.94	30.75	248	1.06	34.05	221	0.95	30.30	269	1.15	34.08
	4	143	0.61	29.50	184	0.79	30.61	221	0.95	33.52	197	0.85	29.43	244	1.05	33.69
64	1	551	2.37	30.43	638	2.74	30.80	658	2.82	34.42	637	2.73	30.80	682	2.92	34.51
	2	440	1.89	30.43	533	2.29	30.80	555	2.38	34.42	538	2.31	30.78	589	2.52	34.50
	4	415	1.78	30.43	500	2.15	30.80	530	2.27	34.42	516	2.22	30.77	567	2.43	34.49
128	1	896	3.85	30.43	959	4.12	30.80	971	4.16	34.42	953	4.09	30.83	987	4.23	34.51
	2	848	3.64	30.43	934	4.01	30.80	948	4.06	34.42	923	3.96	30.83	971	4.16	34.51
	4	839	3.60	30.43	926	3.98	30.80	941	4.03	34.42	914	3.92	30.83	965	4.13	34.51

TABLE 5: The Performance of the Hardware-based BLUT. (N=1K)

We also observed that images with a larger codebook are particularly suitable for BLUT. For example, Table 3 shows that the 2-bitmap BLUT scheme could derive good image quality with D=32. In Tables 4 and 5, the image quality is further improved while the number of calculated codewords is proportional to the size of codebooks.

From our experimental results, one can also notice that the images with better compression quality can benefit more from BLUT, since the maximum one-dimensional distance within these images could be smaller. For example, the compression quality of the images “Lena” and “Zelda” is better than other images. For these two images, the calculated codewords are more than those of other images at the same distance. Therefore, a well-trained codebook could improve the performance of the proposed scheme.

In summary, the performance of BLUT ties to the size of codebooks and its quality. Thus, BLUT is suitable for compressing high-definition images. We also designed a codebook training algorithm for BLUT by minimizing the deviation of the one-dimensional distance between the image blocks and codewords. As compared with the existing schemes, using a feasible hardware implementation without complex computation makes the proposed scheme suitable for digital home entertainment.

4. CONSLUSION & FUTURE WORK

In this study, we propose a new algorithm BLUT for codebook search. BLUT adopts bitwise data structure to represent the geometrical information. The bitwise representation is simple and suitable for hardware implementation. By setting a given range, BLUT could screen any unfeasible codewords. With the software implementation, only one bitmap is required, thereby minimizing the required storage. On the other hand, the hardware implementation in BLUT could adopt multiple bitmaps to ensure accuracy. Since BLUT relies on an accurate geometrical relation to prune eligible codewords, it is suitable for high-definition image compression. In the future, we will gear toward the design of codebook training algorithms to further improve the lookup performance of schemes like BLUT.

5. REFERENCES

1. R. M. Gray, “*Vector Quantization*”, IEEE ASSP Magazine, 1(2): 4-29, 1984

2. A. Gersho, R. M. Gray. "*Vector Quantization and Signal Compression*", Kluwer (1992)

3. T. S. Chen, C. C. Chang. "*An efficient computation of Euclidean distances using approximated look-up table*". IEEE Transactions on Circuits System Video Technology, 10(4): 594-599, 2000

4. C. C. Chang, Y. C. Hu. "*A fast LBG codebook training algorithm for vector quantization*". IEEE Transactions on Consumer Electronics, 44(4):1201-1208, 1988

5. G. A. Davidson, P. R. Cappello and A. Gersho. "*Systolic architectures for vector quantization*". IEEE Transactions on Acoust., Speech, Signal Processing, 36(10):1651-1664, 1988

6. H. Park, V. K. Prasana. "*Modular VLSI architectures for real-time full-search-based vector quantization*". IEEE Transactions on Circuits System Video Technology, 3(4):309-317, 1993

7. P. A. Ramamoorthy, B. Potu and T. Tran. "*Bit-serial VLSI implementation of vector quantizer for real-time image coding*". IEEE Transactions on Circuits System, 36(10):1281-1290, 1989

8. S. A. Rizvi, N. M. Nasrabadi. "*An efficient Euclidean distance computation for quantization using a truncated look-up table*". IEEE Transactions on Circuits System Video Technology, 5(4):370-371, 1995

9. P. Y. Chen, R. D. Chen. "*An index coding algorithm for image vector quantization*". IEEE Transactions on Consumer Electronics, vol. 49, no. 4, pp. 1513-1520, Nov. 2003

10. Singara singh , R. K. Sharma and M.K. Sharma. "*Use of Wavelet Transform Extension for Graphics Image Compression using JPEG2000 Framework*". International Journal of Image Processing, 3(1): 55-60, 2009.

11. W. S. Chen, F. C. Ou, L. C. Lin and C. Hsin. "*Image coding using vector quantization with a hierarchical codebook in wavelet domain*". IEEE Transactions on Consumer Electronics, 45(1):36-45, 1999

12. Y. C. Hu, C. C. Chang. "*Variable rate vector quantization scheme based on quadtree segmentation*". IEEE Trans. Consumer Electronics, 45(2):310-317, 1999

13. R. C. Chen, C. T. Chan, P. C. Wang, T. S. Chen and H. Y. Chang. "*Reducing computation for vector quantization by using bit-mapped look-up table*". In Proceedings of IEEE ICNSC'2004. Taipei, Taiwan, 2004

14. C. C. Chang, C. C. Chen. "*Full-Searching-Equivalent Vector Quantization Method Using Two-Bounds Triangle Inequality*". Fundamenta Informaticae, 76(1-2):25-37, 2007.

15. Rohmad Fakeh, Abdul Azim Abd Ghani. "*Empirical Evaluation of Decomposition Strategy for Wavelet Video Compression*". International Journal of Image Processing, 3(1): 31-54, 2009.

16. C. C. Chang, C. L. Kuo and C. C. Chen. "*Three Improved Codebook Searching Algorithms for Image Compression Using Vector Quantizer*". International Journal of Computers and Applications, 31(1):16-22, 2009.

17. C. C. Chang, W. C. Wu. "Fast Planar-Oriented Ripple Search Algorithm for Hyperspace VQ Codebook". IEEE Transactions on Image Processing, 16(6):1538-1547, 2007.

18. Hiremath P.S, Jagadeesh Pujari. “*Content Based Image Retrieval using Color Boosted Salient Points and Shape features of an image*”. International Journal of Image Processing, 2(1): 10-17, 2008.