Fast Vector Quantization by Using Gated Look-Up Table

Hung-Yi Chang
leorean@isu.edu.tw
Department of Information Management
I-Shou University
Kaohsiung, Taiwan 840, ROC

Pi-Chung Wang, Chia-Tai Chan
{abu, ctchan}@cht.com.tw
Telecommunication Laboratories
Chunghwa Telecom Co., Ltd.
Taipei, Taiwan 106, ROC

Tung-Shou Chen
tschen@ntit.edu.tw
Department of Information Management
National Taichung Institute of Technology
Taichung, Taiwan 404, ROC

Abstract

Vector quantization (VQ) is an elementary technique for image compression. However, the complexity of searching the nearest codeword in a codebook is time-consuming. In this work, we present an adaptive scheme "*Gated Look-Up Table*" (GLUT) which is able to prune codewords rapidly and uses a positional information to represent the geometric relation within codewords. The lookup procedure could jump to the entry codeword and sift candidate codewords easily. This scheme might also cooperate with existing schemes to fasten search speed. The simulation results demonstrate this effectiveness of GLUT.

Keywords: Image compression, nearest neighbor search, vector quantization, look-up table.

1 Introduction

Currently, images have been widely used in computer communications. The sizes of images are usually huge and need to be compressed efficiently for storage and transmission. Vector quantization (VQ) is an important technique for image compression, and has been proven to be simple and efficient [1]. VQ can be defined as a mapping from k-dimensional Euclidean space into a finite subset C of R^k. 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 ($k = w \times h$), then each block is a k-dimensional vector. VQ selects an appropriate 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 jth-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.,

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

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 $[x_{(j)} - c_{i(j)}]^2$s. Since the multiplication is a complex operation, it leads to the increase of the degree of the total computational complexity of equation (1). Therefore, speeding up the calculation of the squared Euclidean distance is a major hurdle.

Many methods have been proposed recently to shorten VQ encoding time [2–6]. The simplest one among them is the *look-up table* (LUT) method [4]. It suggests that the results of the $[x_{(j)} - c_{i(j)}]^2$s 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 L should be $m \times m$ and

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

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

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

LUT can be employed to avoid the subtractions and the multiplications in equation (1). Hence, it is an efficient method.

Rizvi *et. al.* proposed the other LUT-based scheme in [6] to fasten the calculation of squared Euclidean distances, namely, *truncated look-up table* (TLUT) methods. In their first method, Rizvi *et. al.* pre-computed the results of the $(x_{(j)} - c_{i(j)})^2$s for all possible $|x_{(j)} - c_{i(j)}|$, and stored them into matrix T_1. Here

$$T_1 = [0, 1^2, 2^2, \cdots, (m-1)^2]. \tag{4}$$

Since the calculation complexity of 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 T_1, equation (1) can be expressed as follows:

$$d(x, c_i) = \sum_{j=0}^{k-1} T_1[|x_{(j)} - c_{i(j)}|]. \tag{5}$$

The size of matrix T_1 is m. It is still too large for some special designs, such as VLSI implementations and systolic architectures [3, 5].

As a result, Rizvi *et. al.* proposed another method to further reduce the extra storage [6], which is the second TLUT method. In this method, Rizvi *et. al.* pre-computed the matrix T_2 with $T_2[i] = (i \times r)^2$, where $i = 0, 1, 2, \cdots, m/r - 1$, and r is a value of 2 to the power of an integer. Therefore, matrix T_2 can be shown as

$$T_2 = [0, (r \times 1)^2, (r \times 2)^2, \cdots, (r \times (m/r - 1))^2]. \tag{6}$$

Based on the matrix, equation (5) can be rewritten as follows:

$$d(x, c_i) = \sum_{j=0}^{k-1} T_2[\lfloor |x_{(j)} - c_{i(j)}| / r \rfloor]. \tag{7}$$

where the elements of T_2 are numbered starting from zero. Note that r is the reducing factor of this method. The size of matrix T_1 is r times bigger than that of matrix T_2. Thus, this method is more suitable for VLSI implementation. In addition, since r is a value of 2 to the power of an integer, we can employ bit shifting instead of the division operation used in (7). Hence, the second TLUT method employs one absolute subtraction, one shift operation and one memory access in matrix T_2 to achieve the computation of $[x_{(j)} - c_{i(j)}]^2$. The second TLUT method is superior to other methods in terms of its smaller memory usage, but the image quality is poorer. This is because matrix T_2 cannot keep all of the possible squared numbers. As a result, the second TLUT method is prone to making some

mistakes when computing the calculations of the squared Euclidean distances.

As discussed above, the designed criteria of LUT-based schemes emphasize computation speed, table storage and image quality. However, the number of calculated codewords has not mentioned since these schemes did not utilize the geometrical information implied in the codewords. In this work, we propose an adaptive scheme "*Gated Look-Up Table*" (GLUT) which reduces the computed codewords. The new scheme uses two integers to represent the geometric relation within codewords. Accordingly, the search procedure could refer the integers to sift candidate codewords easily. The scheme might also cooperate with existing schemes to fasten the search speed. Simulation results demonstrate this effectiveness. 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 GLUT Method

The VQ selects the codeword with the smallest Euclidean distance to the tested vector. Intuitively, 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 tested vector and the selected codeword should be very close. Hence it is possible to filter out unfeasible codewords by referring the positional information.

We use a two-dimensional case as an example. There are two codewords, C_1 (3, 1) and C_2 (2, 3), as shown in Fig. 1. To calculate the nearest codeword for the tested vector, V_1 (1, 2), the squared Euclidean distances to C_1 and C_2 are 5 and 2, respectively. Hence C_2 should be chosen as the result. If we consider the distances in the vertical axis, C_1 and C_2 will be selected. Thus the Euclidean distance to these two vectors is calculated as well as the full search. However, only C_1 is chosen for V_2 and the calculation for C_2 could be avoided.

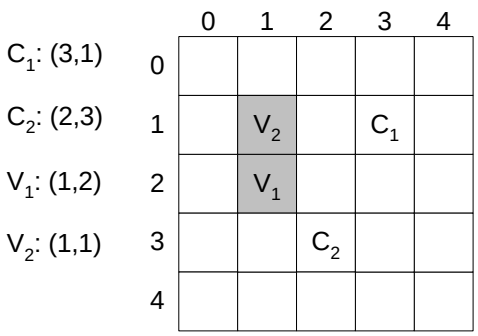

Figure 1. A Two-dimensional Example.

Generally speaking, if the codeword is close to the tested vector in the Euclidean space, the distance in each dimension ought to be short as well. To ease the explanation, we adopt bitmaps to represent the positional information. For each codeword, there is a per-codeword bitmap with m bits for each selected dimension j, where $0 \leq j \leq k-1$. Each bit in the per-codeword bitmap corresponds to a position in dimension j. Assuming that the pre-defined range is D. The bits from $c_{i(j)-D}$ to $c_{i(j)+D}$ is set to one for codeword i.

Figure 2 shows the resulted bitmaps for the example in Fig. 1. The distance D is defined as one. If the tested vector locates in a shadowed brick of a certain codeword, that codeword would be one of the candidates in the vector quantization. For example, V_2 is located within the shadowed bricks of C_1, rather than C_2. Thus C_2 will not be considered in the vector quantization. The bricks at third row are shared by squares of C_1 and C_2. Therefore, the tested vectors positing in these bricks will select C_1 and C_2 as candidates.

In some conditions, there are several unoccupied bricks remaining. For the tested vectors located within these bricks, the bitmaps are useless since there is no candidates could be derived. As a result, each codeword has to be calculated to decide the one with the smallest Euclidean distance. To nail down the problem, a wider range

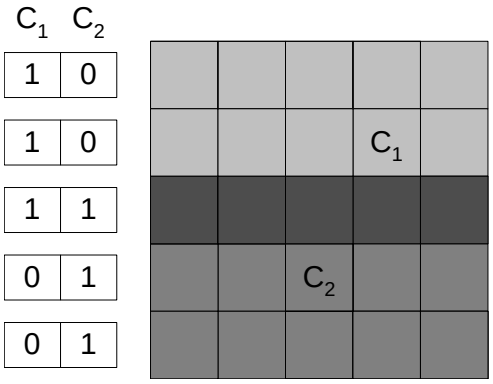

Figure 2. Per-Codeword Bitmaps. ($D = 1$)

could be adopted. However, the conjunct bricks are also increased due to the larger squares, as shown in Fig. 3.

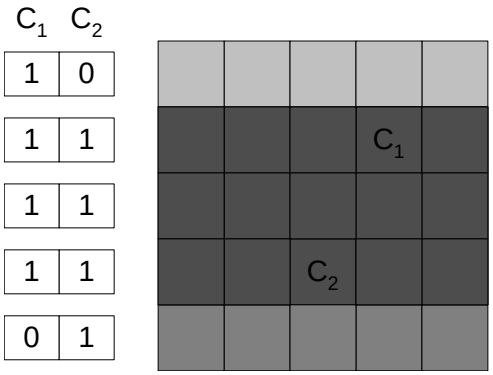

Figure 3. Per-Codeword Bitmaps. ($D = 2$))

A suitable range is thus critical to the performance of the proposed scheme since a wider range could increase the number of candidates while a narrow range might cause null set. In our experiments, various ranges are investigated to evaluate the performance and the image quality. Consequently, the construction/lookup procedure of the searchable data structure "GLUT" is introduced.

2.1 The Construction of the Searchable Data Structure - Gated Look-up Table

Though the per-codeword bitmaps could present the positional information, they are not searchable. This is because accessing bitmaps for each codeword is inefficient. To fasten the lookup speed, the gated look-up table are proposed. Before the gated look-up table is constructed, the intermediate data structure "per-position bitmap" is introduced, as shown in Fig. 4

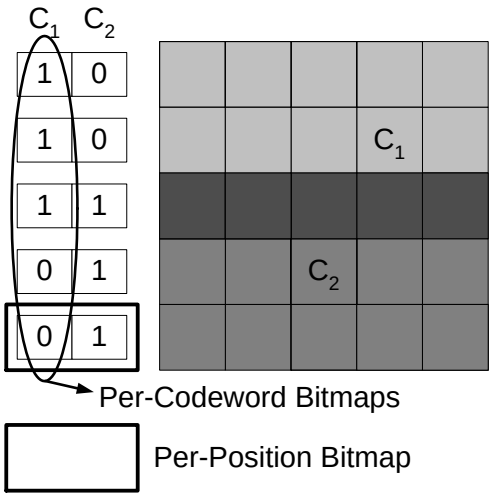

Figure 4. Relationships Between Per-Codeword Bitmaps and Per-Position Bitmaps. ($D = 1$)

The per-position bitmap for position p at dimension j is defined as $B^D_{j,p}$, where D is the preset range. The i_{th} bit is defined as $B^D_{j,p}(i)$ which is set to one if $p - D \leq c_{i(j)} \leq p + D$. The pseudo code is given below. For each range D, the required storage is $m \times N$ bits per dimension. With a typical 256-entry codebook and 256 gray levels, the occupied memory is 8 Kbytes.

Bitmap-Filling Algorithm

For each per-position bitmap $B^D_{j,p}$ **BEGIN**
  For each bit $B^D_{j,p}(i)$ **BEGIN**
    If $p - D \leq c_{i(j)} \leq p + D$, $B^D_{j,p}(i) = 1$.
    Otherwise, $B^D_{j,p}(i) = 0$.
  END
END

The per-position bitmap is simple, but it is not storage-efficient. For a specific dimension and range, the size of the per-position bitmap is 8 Kbytes. Furthermore, the per-position bitmap is suitable for hardware implementation by using a wider memory bus.

Table 1. The Performance of GLUT.

Lena	VQ	TLUT	GLUT Scheme				
			D=16	D=32	D=48	D=64	D=128
PSNR	32.563	32.563	31.294	32.347	32.531	32.554	32.563
Execution Time (sec.)	1.372	1.041	0.201	0.371	0.52	0.671	1.011
Calculated Codewords	256	256	48	92	130	164	243
Memory Size (byte)	0	256	512 (GLUT) + 256 (TLUT)				

Consequently, we address how to use the GLUT to reduce the required storage.

Before executing the bitmap-filling algorithm, the codewords are sorted according to j_{th} value for the selected dimension j. The sequence could be descending or ascending order. In the following, the bitmap-filling algorithm will construct the bitmap in the form $(00\ldots011\ldots1100\ldots00)$ where 1's are consecutive. This is because the sorted codewords also occupy ordered set bits. Hence the first and the last set bits could be recorded as two integers and construct the gated look-up table. For each position p, each element of the GLUT consists of G_p^S and G_p^E which express the number of the first and last set bits. With the simplified representation, the GLUT could achieve fast vector quantization.

The lookup procedure combines fast pruning by GLUT and fast processing by TLUT. For the quantized vector x, the x_jth entry of the GLUT, $G_{x_j}^S$ and $G_{x_j}^E$, is fetched. Each codeword whose index i satisfies $G_{x_j}^S \leq i \leq G_{x_j}^E$ will be selected as the candidate, and the Euclidean distance is calculated by coupling TLUT. The pseudo code for lookup procedure is listed below.

VQ Algorithm by GLUT
For each vector x **BEGIN**
  Fetch the $G_{x_j}^S$ and $G_{x_j}^E$.
  For each codeword i, where $G_{x_j}^S \leq i \leq G_{x_j}^E$
  BEGIN
    Calculate Euclidean distance $d(x, c_i)$
    $d(x, c_i) = \sum_{j=0}^{k-1} TLUT_1[|x_{(j)}, c_{i(j)}|]$.
    If $d(x, c_i) \geq max\_distance$ **BEGIN**
      $max\_distance\_id = i$
      $max\_distance = d(x, c_i)$
    END
  END
  $max\_distance\_id$ is the result for x.
END

3 Simulation Results

We have conducted several simulations to show the efficiency of GLUT. 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 [1,2], 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 are estimated by the *peak signal-to-noise ratio* (PSNR). A larger PSNR value indicates better preserved the original image quality. The extra memory denotes the storage needed to record the GLUT and TLUT. As for the mathematical operations, the number of calculated codewords is considered since the operations for each codeword are identical.

The experiments were performed on an IBM PC with a 500-MHz Pentium CPU. Table 1 shows the experimental results of GLUT. VQ indicates the vector quantiza-

(a) D = 16 (PSNR=31.294) (b) D = 32 (PSNR=32.347) (c) D = 64 (PSNR=32.554)

Figure 5. The Decompressed Lena Images of GLUT Scheme.

tion without any speedup. The ranges for GLUT vary from 16 to 128. With a smaller range, the image quality is degraded since the occurrence of false matches is increased. Nevertheless, the calculated codewords are reduced by per-position bitmap, the execution time is lessened as well. VQ requires no extra storage while the TLUT needs 256 bytes. The extra storage is 512 bytes for GLUT and 256 bytes for TLUT. The decompressed images are shown in Fig. 5. While GLUT range is enlarged to 64, the image quality is almost identical to VQ and TLUT.

4 Conclusion

In this study, we propose a new algorithm "GLUT" for fast codebook search. The GLUT adopts simple data structure with merely two integers to represent the geometrical information. By setting a given range, the GLUT could sift out codewords easily, and the GLUT is suitable for software implementation. The extra storage is 512 bytes and the performance could be improved with a factor of five. In the future, we will apply this concept to the codebook training.

References

[1] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression.* Boston, MA: Kluwer, 1992.

[2] T. S. Chen and C. C. Chang, "An Efficient Computation of Euclidean Distances Using Approximated Look-Up Table," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 7, pp. 594-599, June 2000.

[3] G. A Davidson, P. R. Cappello and A. Gersho, "Systolic architectures for vector quantization," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 36, pp. 1651-1664, Oct. 1988.

[4] H. Park and V. K. Prasana, "Modular VLSI architectures for real-time full-search-based vector quantization," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 3, pp. 309-317, Aug. 1993.

[5] P. A. Ramamoorthy, B. Potu and T. Tran, "Bit-serial VLSI implementation nof vector quantizer for real-time image coding ," *IEEE Trans. Circuits Syst.*, vol. 36, pp. 1281-1290, Oct. 1989.

[6] S. A. Rizvi and N. M. Nasrabadi, "An efficient euclidean distance computation for quantization using a truncated look-up table," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 5, pp. 370-371, Aug. 1995.