

## Research Article

# A New Multistage Lattice Vector Quantization with Adaptive Subband Thresholding for Image Compression

M. F. M. Salleh and J. Soraghan

*Institute for Signal Processing and Communications, Department of Electronic and Electrical Engineering,  
University of Strathclyde, Royal College Building, Glasgow G1 1XW, UK*

Received 22 December 2005; Revised 2 December 2006; Accepted 2 February 2007

Recommended by Liang-Gee Chen

Lattice vector quantization (LVQ) reduces coding complexity and computation due to its regular structure. A new multistage LVQ (MLVQ) using an adaptive subband thresholding technique is presented and applied to image compression. The technique concentrates on reducing the quantization error of the quantized vectors by “blowing out” the residual quantization errors with an LVQ scale factor. The significant coefficients of each subband are identified using an optimum adaptive thresholding scheme for each subband. A variable length coding procedure using Golomb codes is used to compress the codebook index which produces a very efficient and fast technique for entropy coding. Experimental results using the MLVQ are shown to be significantly better than JPEG 2000 and the recent VQ techniques for various test images.

Copyright © 2007 M. F. M. Salleh and J. Soraghan. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

## 1. INTRODUCTION

Recently there have been significant efforts in producing efficient image coding algorithms based on the wavelet transform and vector quantization (VQ) [1–4]. In [4], a review of some of image compression schemes that use vector quantization and wavelet transform is given. In [1] a still image compression scheme introduces an adaptive VQ technique. The high frequency subbands coefficients are coded using a technique called multiresolution adaptive vector quantization (MRAVQ). The VQ scheme uses the LBG algorithm wherein the codebook is constructed adaptively from the input data. The MRAVQ uses a bit allocation technique based on marginal analysis, and also incorporates the human visual system properties. MRAVQ technique has been extended to video coding in [5] to form the adaptive joint subband vector quantization (AJVQ). Using the LBG algorithm results in high computation demands and encoding complexity particularly as the vector dimension and bit rate increase [6]. The lattice vector quantization (LVQ) offers substantial reduction in computational load and design complexity due to the lattice regular structure [7]. The LVQ has been used in many image coding applications [2, 3, 6]. In [2] a multistage residual vector quantization based on [8] is used along with LVQ

that produced results that are comparable to JPEG 2000 [9] at low bit rates.

Image compression schemes that use plain lattice VQ have been presented in [3, 6]. In order to improve performance, the concept of *zerotree* prediction as in EZW [10] or SPHIT [11] is incorporated to the coding scheme as presented in [12]. In this work the authors introduce a technique called vector-SPHIT (VSPHIT) that groups the wavelet coefficients to form vectors before using *zerotree* prediction. In addition, the significant coefficients are quantized using the voronoi lattice VQ (VLVQ) that reduces computational load. Besides scanning the individual wavelet coefficients based on *zerotree* concept, scanning blocks of the wavelet coefficients has recently become popular. Such work is presented in [13] called the “set-partitioning embedded block” (SPECK). The work exploits the energy cluster of a block within the subband and the significant coefficients are coded using a simple scalar quantization. The work in [14] uses VQ to code the significant coefficients for SPECK called the vector SPECK (VSPECK) which improves the performance.

The image coding scheme based on the wavelet transform and vector quantization in [1, 2] searches for the significant subband coefficients by comparing them to a threshold value at the initial compression stage. This is followed by a

quadtree modelling process of the significant data location. The threshold setting is an important entity in searching for the significant vectors in the subbands. Image subbands at different levels of decomposition carry different degrees of information. For general images, lower frequency subbands carry more significant data than higher frequency subbands [15]. Therefore there is a need to optimize the threshold values for each subband. A second level of compression is achieved by quantizing the significant vectors.

Entropy coding or lossless coding is traditionally the last stage in an image compression scheme. The run-length coding technique is very popular choice for lossless coding. Reference [16] reports an efficient entropy coding technique for sequences with significant runs of zeros. The scheme is used on test data compression for a system-on-a-chip design. The scheme incorporates variable run-length coding and Golomb codes [17] which provide a unique binary representation for run-length integer symbol with different lengths. It also offers a fast decoding algorithm as reported in [18].

In this paper, a new technique for searching the significant subband coefficients based on an adaptive thresholding scheme is presented. A new multistage LVQ (MLVQ) procedure is developed that effectively reduces the quantization errors of the quantized significant data. This is achieved as a result of having a few quantizers in series in the encoding algorithm. The first quantizer output represents the quantized vectors and the remaining quantizers deal with the quantization errors. For stage 2 and above the quantization errors are “blown out” using an LVQ scale factor. This allows the LVQ to be used more efficiently. This differs from [2] wherein the quantization errors are quantized until the residual quantization errors converge to zero. Finally the variable length coding with the Golomb codes is employed for lossless compression of the lattice codebook index data.

The paper is organized as follows. Section 2 gives a review of Golomb coding for lossless data compression. Section 3 reviews basic vector quantization and the lattice VQ. The new multistage LVQ (MLVQ), adaptive subband thresholding algorithm, and the index compression technique based on Golomb coding are presented in Section 4. The performance of the multiscale MLVQ algorithm for image compression is presented in Section 5. MLVQ is shown to be significantly superior to Man’s [2] method and JPEG 2000 [9]. It is also better than some recent VQ works as presented in [12, 14]. Section 6 concludes the paper.

## 2. GOLOMB CODING

In this section, we review the Golomb coding and its application to binary data having long runs of zeros. The Golomb code provides a variable length code of the integer symbol [17]. It is characterized by the Golomb code parameter  $b$  which refers to the group size of the code. The choice of the optimum value  $b$  for a certain data distribution is a non-trivial task. An optimum value of  $b$  for random distribution of binary data has been found by Golomb [17] as follows.

Consider a sequence of length  $N$  having  $n$  zeros and a one  $\{00 \dots 01\}$

$$X = \{0^n 1\}; \quad \text{where } N = n + 1. \quad (1)$$

Let  $p$  be the probability of a zero, and  $1 - p$  is the probability of a one

$$P(0) = p, \quad P(1) = 1 - p. \quad (2)$$

The probability of the sequence  $X$  can be expressed as

$$P(n) = p^n(1 - p). \quad (3)$$

The optimum value of the group size  $b$  is [12]

$$b = \left\lceil -\frac{1}{\log_2 p} \right\rceil. \quad (4)$$

The run-length integers are grouped together, and the element in the group set is based on the optimum Golomb parameter  $b$  found in (4). The run lengths (integer symbols) group set  $G_1$  is  $\{0, 1, 2, \dots, b - 1\}$ ; the run lengths (integer symbols) group set  $G_2$  is  $\{b, b + 1, b + 2, \dots, 2b - 1\}$ ; and so forth. If  $b$  is a value of the power of two ( $b = 2^N$ ), then each group  $G_k$  will have  $2^N$  number of run lengths (integer symbols). In general, the set of run lengths (integer symbols)  $G_k$  is given by the following [17]:

$$G_k = \{(k - 1)b, (k - 1)b + 1, (k - 1)b + 2, \dots, kb - 1\}. \quad (5)$$

Each group of  $G_k$  will have a prefix and  $b$  number of tails. The prefix is denoted as  $(k - 1)1$ s followed by a zero defined as

$$\text{prefix} = 1^{(k-1)}0. \quad (6)$$

The tails is a binary representation of the modulus operation between the run length integer symbol and  $b$ . Let  $n$  be the length of tail sequence

$$n = \log_2 b,$$

$$\text{tail} = \text{mod}(\text{run\_length symbol}, b) \text{ with } n \text{ bits length.} \quad (7)$$

The codeword representation of the run length consists of two parts, that is, the prefix and tail. Figure 1 summarized the process of Golomb coding for  $b = 4$ . From (5) the first group will consist of the run-length  $\{0, 1, 2, 3\}$  or  $G_1 = \{0, 1, 2, 3\}$ , and  $G_2 = \{4, 5, 6, 7\}$ , and so forth. Group 1 will have a prefix  $\{0\}$ , group 2 will have prefix  $\{10\}$ , and so forth. Since the value of  $b$  is chosen as 4, the length of tail is  $\log_2 4 = 2$ . For run-length 0, the tail is represented by bits  $\{00\}$ , the tail for run-length 1 is represented by bits  $\{10\}$ , and so forth. Since the codeword is the combination of the group prefix and the tail, for run-length of 0 will have the codeword of  $\{000\}$  where the first 0 is the group prefix and the remaining 0s is the tail, the run-length 1 will have codeword  $\{001\}$ , and so forth. Figure 1(b) shows the example of the encoding process with 32 bits input with 6 ones ( $r = 6$ ) which can be encoded as 22 bits. The Golomb codes offer an efficient technique for run-length coding (variable-length coding).

Group	Run length	Group prefix	Tail	Codeword
G1	0	0	00	000
	1		01	001
	2		10	010
	3		11	011
G2	4	10	00	1000
	5		01	1001
	6		10	1010
	7		11	1011
G3	8	110	00	11000
	9		01	11001
	10		10	11010
	11		11	11011
...	...	...	...	...

(a) Golomb coding for  $b = 4$ 

$$S = \{ \underbrace{000001}_{l_1=5} \underbrace{00001}_{l_2=4} \underbrace{00000001}_{l_3=6} \underbrace{1}_{l_5=0} \underbrace{00000001}_{l_6=7} \}$$

$$\begin{array}{cccccc} l_1 = 5 & l_2 = 4 & l_3 = 6 & l_5 = 0 & l_6 = 7 & \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \\ CS = \{1001 & 1000 & 1011 & 000 & 1011\} \end{array}$$

(b) Example of encoding using the Golomb code  $b = 4$ , CS = 19 bits

FIGURE 1

### 3. VECTOR QUANTIZATION

#### 3.1. Lattice vector quantization

Vector quantizers (VQ) maps a cluster of vectors to a single vector or codeword. A collection of codewords is called a codebook. Let  $X$  be an input source vector with  $n$ -components with joint pdf  $f_X(x) = f_X(x_1, x_2, \dots, x_n)$ . A vector quantization is denoted as Q with dimension  $n$  and size  $L$ . It is defined as a function that maps a specific vector  $X \in \mathfrak{X}^n$  into one of the finite sets of output vectors of size  $L$  to be  $Y_i = Y_1, Y_2, \dots, Y_L$ . Each of these output vectors is the codeword and  $Y \in \mathfrak{X}^n$ . Around each codeword  $Y_i$ , an associated nearest neighbour set of points called Voronoi regions are defined as [19]

$$V(Y_i) = \{x \in \mathfrak{X}^k : \|x - Y_i\| \leq \|x - Y_j\| \quad \forall i \neq j. \quad (8)$$

In lattice vector quantization (LVQ), the input data is mapped to the lattice points of a certain chosen lattice type. The lattice points or codewords may be selected from the coset points or the truncated lattice points [19]. The coset of a lattice is the set of points obtained after a specific vector is added to each lattice point. The input vectors surrounding these lattice points are grouped together as if they are in the same voronoi region.

The codebook of a lattice quantizer is obtained by selecting a finite number of lattice points (codewords of length  $L$ ) out of infinite lattice points. Gibson and Sayood [20] used the minimum peak energy criteria of a lattice point in choosing

the codewords. The peak energy is defined as the squared distance of an output point (lattice point) farthest from the origin. This rule dictates the filling order of  $L$  codewords starting from the innermost shells. The number of lattice point on each shell is obtained from the coefficient of the theta function [7, 20]. Sloane has tabulated the number of lattice points in the innermost shells of several root lattices and their dual [21].

#### 3.2. Lattice type

A lattice is a regular arrangement of points in  $k$ -space that includes the origin or the zero vector. A lattice is defined as a set of linearly independent vectors [7];

$$\Lambda = \{X : X = a_1 u_1 + a_2 u_2 + \dots + a_N u_N\}, \quad (9)$$

where  $\Lambda \in \mathfrak{R}^k$ ,  $n \leq k$ ,  $a_i$  and  $u_i$  are integers for  $i = 1, 2, \dots, N$ . The vector set  $\{u_i\}$  is called the basis vectors of lattice  $\Lambda$ , and it is convenient to express them as a generating matrix  $U = [u_1, u_2, \dots, u_n]$ .

The  $Z^n$  or cubic lattice is the simplest form of a lattice structure. It consists of all the points in the coordinate system with a certain lattice dimension. Other lattices such as  $D_n (n \geq 2)$ ,  $A_n (n \geq 1)$ ,  $E_n [n = 6, 7, 8]$ , and their dual are the densest known sphere packing and covering in dimension  $n \leq 8$  [16]. Thus, they can be used for an efficient lattice vector quantizer. The  $D_n$  lattice is defined by the following [7]:

$$D_n = (x_1, x_2, \dots, x_n) \in Z^n, \quad \text{where } \sum_{i=1}^n x_i = \text{even}. \quad (10)$$

The  $A_n$  lattice for  $n \geq 1$  consists of the points of  $(x_0, x_1, \dots, x_n)$  with the integer coordinates sum to zero. The lattice quantization for  $A_n$  is done in  $n + 1$  dimensions and the final result is obtained after reverting the dimension back to  $n$ . The expression for  $E_n$  lattice with  $n = 6, 7, 8$  is explained in [7] as the following:

$$E_8 = \left( \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2} \right) + D_8. \quad (11)$$

The dual of lattice  $D_n$ ,  $A_n$ , and  $E_n$  are detailed in [7]. Besides, other important lattices have also been considered for many applications such as the Coxeter-Todd ( $K_{12}$ ) lattice, Barnes-Wall lattice ( $\Lambda_{16}$ ), and Leech lattice ( $\Lambda_{24}$ ). These lattices are the densest known sphere packing and coverings in their respective dimension [7].

#### 3.3. Quantizing algorithms

Quantizing algorithms were developed based on the knowledge of the root lattices and their dual for finding the closest lattice point to an arbitrary point  $x$  in the space. Conway and Sloane [22] developed an algorithm for finding the closest point of the  $n$ -dimensional integer lattice  $Z^n$ . The  $Z^n$  or cubic lattice is the simplest form of a lattice structure and thus finding the closest point in the  $Z^n$  lattice to the arbitrary point or input vectors in space  $x \in \mathfrak{X}^n$  is straightforward.

Define  $f(x) = \text{round}(x)$  and  $w(x)$  as

$$\begin{aligned} w(x) &= \lceil x \rceil && \text{for } 0 < x < 0.5 \\ &= \lfloor x \rfloor && \text{for } x > 0.5 \\ &= \lfloor x \rfloor && \text{for } -0.5 < x \leq 0 \\ &= \lceil x \rceil && \text{for } x < -0.5, \end{aligned} \quad (12)$$

where  $\lfloor \cdot \rfloor$  and  $\lceil \cdot \rceil$  are the floor and ceiling functions, respectively.

The following sequences give the clear representation of the algorithm where  $u$  is an integer.

- (1) If  $x = 0$ , then  $f(x) = 0$ ,  $w(x) = 1$ .
- (2) If  $-1/2 \leq x < 0$  then  $f(x) = 0$ ,  $w(x) = -1$ .
- (3) If  $0 < x < 1/2$ ,  $u = 0$  then  $f(x) = u$ ,  $w(x) = u + 1$ .
- (4) If  $0 < u \leq x \leq u + 1/2$ , then  $f(x) = u$ ,  $w(x) = u + 1$ .
- (5) If  $0 < u + 1/2 < x < u + 1$ , then  $f(x) = u + 1$ ,  $w(x) = u$ .
- (6) If  $-u - 1/2 \leq x \leq -u < 0$ , then  $f(x) = -u$ ,  $w(x) = -u - 1$ .
- (7) If  $-u - 1 < x < -u - 1/2$ , then  $f(x) = -u - 1$ ,  $w(x) = -u - 1/2$ .

Conway and Sloane [22] also developed quantizing algorithms for other lattices such as the  $D_n$ , which is the subset of lattice  $Z^n$  and  $A_n$ . The  $D_n$  lattice is formed after taking the alternate points of the  $Z^n$  cubic lattice [7]. For a given  $x \in \mathfrak{R}^n$  we define  $f(x)$  as the closest integer to input vector  $x$ , and  $g(x)$  is the next closest integer to  $x$ . The sum of all components in  $f(x)$  and  $g(x)$  is obtained. The quantizing output is chosen from either  $f(x)$  or  $g(x)$  whichever has an even sum [22]. The algorithm for finding the closest point of  $A_n$  to input vector or point  $x$  has been developed by Conway and Sloane, and is given by the procedure defined in [22]. The quantization process will end up with the chosen lattice points to form a hexagonal shape for two dimensional vectors.

## 4. A NEW MULTISTAGE LATTICE VQ FOR IMAGE COMPRESSION

### 4.1. Image encoder architecture

Figure 2 illustrates the encoder part of the new multiscale-based multistage LVQ (MLVQ) using adaptive subband thresholding and index compression with Golomb codes. A wavelet transform is used to transform the image into a number of levels. A vector or unit is obtained by subdividing the subband coefficients into certain block sizes. For example, a block size of  $4 \times 4$  gives a 16 dimensional vector,  $2 \times 2$  gives 4 dimensional vector, and  $1 \times 1$  gives one dimensional vector. The significant vectors or units of all subbands are identified by comparing the vector energy to certain thresholds. The location information of the significant vectors is represented in ones and zeros, defined as a MAP sequence which is coded using quadtree coding. The significant vectors are saved and passed to the multistage LVQ (MLVQ). The MLVQ produces two outputs, that is, the scale list and index sequence, which are then run-length coded. The lowest frequency subband is coded using the JPEG 2000 lossless coding. The details of

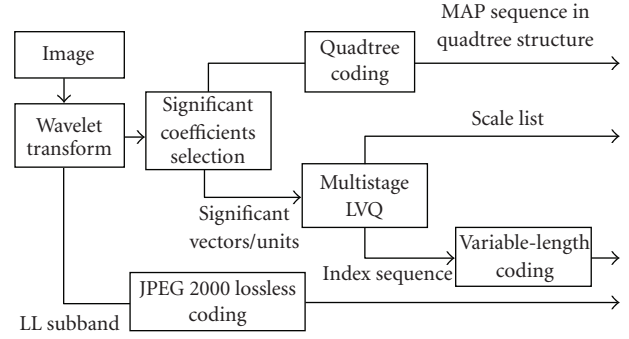


FIGURE 2: MLVQ encoder scheme.

MLVQ and the generation of  $M$ -stage codebook for a particular subband are described in Section 4.3.

### 4.2. Adaptive subband thresholding

The threshold setting is an important entity in searching for the significant coefficients (vectors/units) in the subband. A vector or unit which consists of the subband coefficients is considered significant if its normalized energy  $E$  defined as

$$E = \frac{w(k)}{N_k x N_k} \sum_{i=1}^{N_k} \sum_{j=1}^{N_k} (X_k(i, j))^2 \quad (13)$$

is greater than a threshold  $T$  defined as

$$T = \left( \frac{\text{av}}{100} \times \text{threshold parameter} \right)^2, \quad (14)$$

where  $X_k$  is a vector in a particular subband  $k$  with dimension  $N_k$ ,  $w(k)$  is the perceptual weight factor and  $\text{av}$  is the average pixel value of input image. The “threshold parameter” which has a valid value of 1 to 1000, is chosen by taking into account the target bit rate. Image subbands at different levels of decomposition carry different weights of information. The lower frequency subbands carry more significant data as compared to the higher ones [15]. Also different subbands at the same wavelet transform level have different statistical distributions. Thus, we introduce an adaptive subband thresholding scheme, which adapts the threshold values in two steps. First the scheme optimizes the threshold values between the wavelet transform levels. Then, these threshold values are optimized at each wavelet transform level. In both steps, the threshold values are optimized by minimizing the distortion of the reconstructed image. The process is also restricted by a bit allocation constraint. In this case the bit allocation was bounded using the amount of vectors available (15). We define  $R$  as the target bit rate per pixel (bpp),  $r$  and  $c$  are the number of row and column of the image, and  $\text{LL\_sb\_bit}$  is the amount of bits required to code the low-low subband and other  $\text{sb\_bits}$  is the amount of bits required to

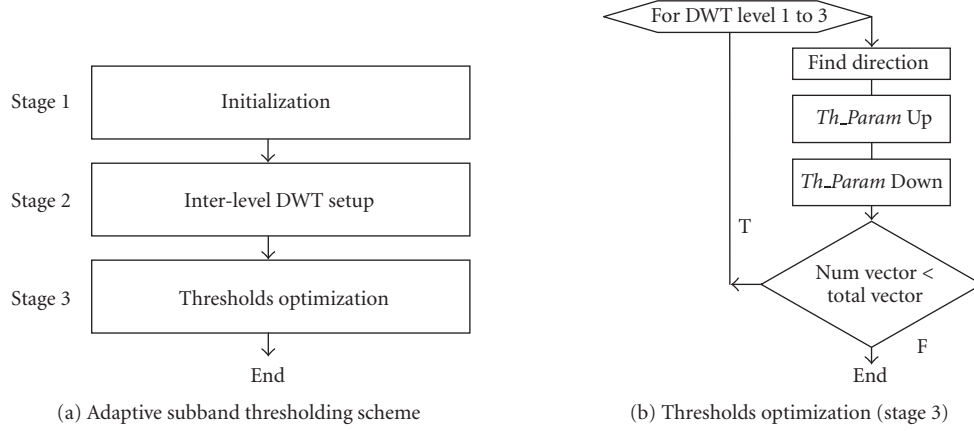


FIGURE 3

code the remaining subbands and  $\text{bit}_{\text{budget}}$  is the total bit budget. The following relationships are defined:

$$\begin{aligned} \text{bit}_{\text{budget}} &= R \times (r \times c) = \text{LL sb\_bits} + \text{other sb\_bits}, \\ \text{total\_no\_vectors} &= \sum_{i=1}^{L_{\max}} \frac{(\text{other sb\_bits} - 0.2 \times \text{other sb\_bits} - 3 \times 8)}{\rho} \end{aligned} \quad (15)$$

where  $\rho = \begin{cases} 6, & n = 4, \\ 3, & n = 1. \end{cases}$

In this work the wavelet transform level ( $L_{\max}$ ) is 3, and we are approximating 20% of the high-frequency subband bits to be used to code the MAP data. For  $Z^n$  lattice quantizer with codebook radius ( $m = 3$ ), the denominator  $\rho$  is 6 (6-bit index) for  $n = 4$  or 3 (3-bit index) for  $n = 1$ . The last term in (15) accounts for the LVQ scale factors, where there are 3 high-frequency subbands available at every wavelet transform level, and each of the scale factors is represented by 8 bits.

The adaptive threshold algorithm can be categorized into three stages as shown by the flow diagram in Figure 3(a). The first stage (initialization) calculates the initial threshold using (14), and this value is used to search the significant coefficients in the subbands. Then the sifted subbands are used to reconstruct the image and the initial distortion is calculated. In the second stage (inter-level DWT setup) the algorithm optimizes the threshold between the wavelet transform levels. Thus in the case of a 3-level system there will be three threshold values for the three different wavelet levels.

An iterative process is carried out to search for the optimal threshold setup between the wavelet transform levels. The following empirical relationship between threshold values at different levels is used:

$$T_l = \begin{cases} T_{\text{initial}}, & \text{for } l = 1, \\ \frac{T_{\text{initial}}}{(l-1) \times \delta}, & \text{for } l > 1, \end{cases} \quad (16)$$

where  $T_{\text{initial}}$  is the initial threshold value,  $T_l$  indicates the threshold value at DWT level  $l$ , and  $\delta$  is an incremental counter.

In the search process every time the value of  $\delta$  is incremented the above steps are repeated for calculating the distortion and resulting output number of vectors that are used. The process will stop and the optimized threshold values are saved once the current distortion is higher than the previous one.

The third stage (thresholds optimization) optimizes the threshold values for each subband at every wavelet transform level. Thus there will be nine different optimized threshold values. The three threshold values found in stage 2 above are used in subsequent steps for the “threshold parameter” expression derived from (14) as follows:

$$\text{threshold parameter} = \left( \frac{100}{\text{av}} \right) \sqrt{T_l} \quad \text{where } l = 1, 2, 3. \quad (17)$$

In this stage the algorithm optimizes the threshold by increasing or lowering the “threshold parameter.” The detail flow diagram of the threshold optimization process is shown in Figure 3(b).

The first process (find direction) is to identify the direction of the “threshold parameter” whether up or down. Then the (*Th\_Param Up*) algorithm processes the subbands that have the “threshold parameter” going up. In this process, every time the “threshold parameter” value increases, a new threshold for that particular subband is computed. Then it searches the significant coefficients and the sifted subbands are used to reconstruct the image. Also the number of significant vectors within the subbands and resulting distortion are computed. The optimization process will stop, and the optimized values are saved when the current distortion is higher than the previous one or the number of vectors has exceeded the maximum allowed.

Finally, the (*Th\_Param Down*) algorithm processes the subbands which have the “threshold parameter” going down. It involves the same steps as above before calculating the distortion. The vector gain obtained in the above step is used



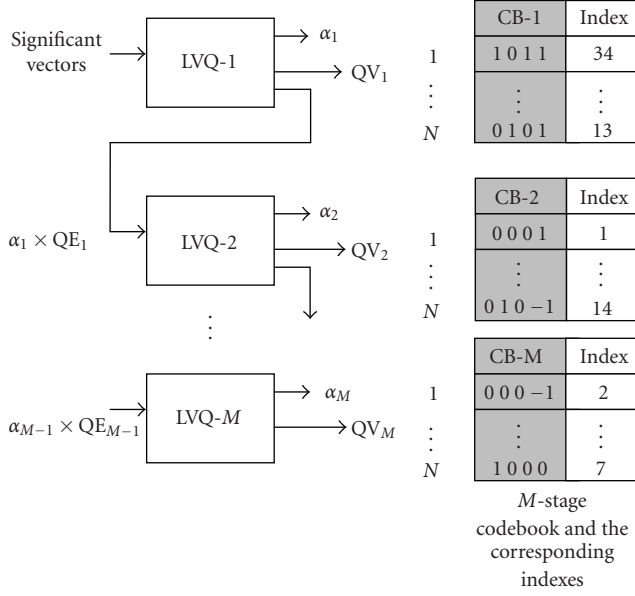


FIGURE 4: MLVQ process of a particular subband.

as the lower bound. The optimized values are saved after the current distortion is higher than the previous one or the number of vectors has exceeded the maximum allowed.

### 4.3. Multistage lattice VQ

The multistage LVQ (MLVQ) process for a particular subband is illustrated in Figure 4. In this paper we chose the  $Z^n$  lattice quantizer to quantize the significant vectors. For each LVQ process, the input vectors are first scaled and then the scaled vectors are quantized using the quantizing algorithm presented in Section 3.3. The output vectors of this algorithm are checked to make sure that they are confined in the chosen spherical codebook radius. The output vectors that exceed the codebook radius are rescaled and remapped to the nearest valid codeword to produce the final quantized vectors (QV). The quantization error vectors are obtained by subtracting the quantized vectors from the scaled vectors. Therefore each LVQ process produces three outputs, that is, the scale factor ( $\alpha$ ), quantized vectors (QV), and the quantization error vectors (QE).

The scaling procedure for each LVQ of the input vectors uses the modification of the work presented in [3]. As a result of these modifications, we can use the optimum setup (obtained from experiment) for codebook truncation where the input vectors reside in both granular and overlap regions for LVQ stage one. At the subsequent LVQ stages the input vectors are forced to reside only in granular regions. The first LVQ stage processes the significant vectors and produces a scale factor ( $\alpha_1$ ), the quantized vectors ( $QV_1$ ) or codewords, and the quantization error vectors ( $QE_1$ ), and so forth. Then the quantization error vectors ( $QE_1$ ) are “blown out” by multiplying them with the current stage scale factor ( $\alpha_1$ ). They are then used as the input vectors for the subsequent LVQ

stage, and this process repeats up to stage  $M$  until the allocated bits are exhausted.

Figure 4 illustrates the resulting  $M$ -stage codebook generation and the corresponding indexes of a particular subband. At each LVQ stage, a spherical  $Z^n$  quantizer with codebook radius ( $m = 3$ ) is used. Hence for four dimensional vectors, there are 64 lattice points (codewords) available with 3 layers codebook [3]. The index of each codeword is represented by 6 bits. If the origin is included, the outer lattice point will be removed to accommodate the origin. In one dimensional vector there are 7 codewords with 3 bits index representation. If a single stage LVQ produces  $N$  codewords and there are  $M$  stages, then the resulting codebook size is  $M \times N$  as shown in Figure 4. The indexes of  $M$ -stage codebook are variable-length coded using the Golomb codes.

The MLVQ pseudo code to process all the high-frequency subbands is described in Figure 5. The  $L_{\max}$  indicates the number of DWT level. In this algorithm, the quantization errors are produced for an extra set of input vectors to be quantized. The advantage of “blowing out” the quantization error vectors is that they can be mapped to many more lattice points during the subsequent LVQ stages. Thus the MLVQ can capture more quantization errors and produce better image quality.

### 4.4. Lattice codebook index compression

Run-length coding is useful in compressing binary data sequence with long runs of zeros. In this technique each run of zeros is represented by integer values or symbols. For example, a 24-bit binary sequence  $\{00000000001000000001\}$  can be encoded as an integer sequence  $\{10, 8\}$ . If each run-length integer is represented by 8-bit, the above sequence can be represented as 16-bit sequence. This method is inefficient when most of the integer symbols can be represented with less than 8-bits or when some of the integer symbols exceed the 8-bit value. This problem is solved using a variable-length coding with Golomb codes, where each integer symbol is represented by a unique bit representation of different Golomb codes length [17].

In our work, we use variable-length coding to compress the index sequence. First we obtained the value of  $b$  as follows assuming that  $X$  is a binary sequence with length  $N$ :

$$P(0) = p; \quad P(1) = 1 - p = \left[ \frac{\sum_{i=1}^N x_i}{N} \right]; \quad x_i \in X. \quad (18)$$

From (4) we can derive the value of  $b$

$$b = \text{round} \left( \left\lceil -\frac{1}{\log_2(1 - \delta)} \right\rceil \right); \quad \text{where } \delta = \left[ \frac{\sum_{i=1}^N x_i}{N} \right]. \quad (19)$$

In this work for 4 dimensional vector, the index sequence consists of integer values with maximum value of 64, and can be represented as 6-bit integer. The distribution of these index values is dependent upon the MLVQ stage. For example, the index data are widely distributed between 1 and 64 (3 codebook levels) at stage one. However, the distribution

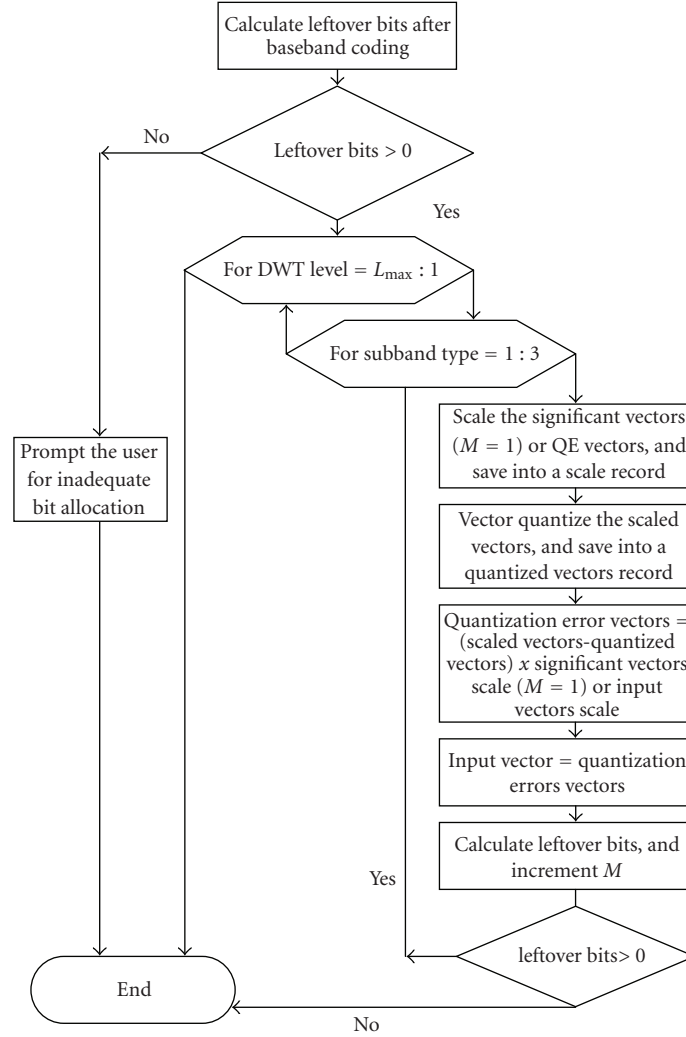


FIGURE 5: Flow diagram of MLVQ algorithm.


is more concentrated on the first codebook level and origin when the multistage is greater than one. This is due to the fact that the MLVQ has been designed to force the quantized vectors to reside in the granular region if the multistage has more than 1 stage as explained in Section 4.3. Figure 6 illustrates the index compression scheme using variable-length coding with Golomb codes for 4-dimensional vector codebook indexes.

The compression technique involves two steps for the case of stage one of the MLVQ. First the index sequence is changed to binary sequence, and then split into two parts, that is, the higher nibble and the lower nibble. The compression is done only on the higher nibble since it has more zeros and less ones. The lower nibble is uncompressed since it has almost 50% zeros and ones. Figure 6 illustrates the index compression technique for MLVQ of stage one. The higher nibble index column bits are taken, and they are jointed together as a single row of bit sequence  $S$ . Then the coded sequence  $CS$  is produced via variable length coding with Golomb codes with parameter  $b = 4$ . From Figure 1(a), the

first run-length ( $l_1 = 9$ ) is coded as  $\{11001\}$ , the second run-length ( $l_2 = 0$ ) is coded as  $\{000\}$ , the third run-length ( $l_3 = 1$ ) is coded as  $\{001\}$ , and so forth. For the subsequent stages for 4-dimensional vector of MLVQ, the entire data will be compressed rather than dividing them into the higher and lower nibbles. For 1 dimensional vector, the codebook indexes are represented as 3-bit integers and the whole binary data are compressed for every MLVQ stage. In this work, the variable length coding with Golomb codes provides high compression on the index sequences. Thus more leftover bits are available for subsequent LVQ stages to encode quantization errors and yield better output quality.

## 5. SIMULATION RESULTS

The test images are decomposed into several WT levels. In this work we used 4 WT levels for image size  $512 \times 512$ , and 3 levels for image size  $256 \times 256$ . Various block sizes are used for truncating the subbands which ultimately determine the vector size. The  $2 \times 2$  block size results in four dimensional



Index	Higher nibble			Lower nibble		
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	0	0	1	0	0
9	0	0	1	0	0	1
5	0	0	0	1	0	1
1	0	0	0	0	0	0
7	0	0	0	1	1	1
15	0	0	1	1	1	1
12	0	0	1	1	0	0
52	1	1	0	1	0	0
40	1	0	1	0	0	0
12	0	0	1	1	0	0
60	1	1	1	1	1	1

$$S = \{\underbrace{0000000001}_{l_1=9} \underbrace{1}_{l_2=0} \underbrace{01}_{l_3=1} 00\dots1111\}$$

$$CS = \{11001 \quad 000 \quad 001\dots\}$$

FIGURE 6: Index sequence compression (multistage = 1).

vectors, and block size  $1 \times 1$  results in one dimensional vector. In this work the block size  $1 \times 1$  is used in the lower subbands with maximum codebook radius set to ( $m = 2$ ). In this case, every pixel can be lattice quantized to one of the following values  $\{0, \pm 1, \pm 2\}$ . Since the lower subbands contain more significant data, there are higher number data being quantized to either to  $\{\pm 2\}$  (highest codebook radius). This increases the codebook index redundancy resulting in a higher overall compression via entropy coding using the variable-length coding with Golomb codes.

### 5.1. Incremental results

In MLVQ the quantization errors of the current stage are “blown out” by multiplying them with the current scale factor. The advantage of “blowing out” the quantization errors is that there will be more lattice points in the subsequent quantization stages. Thus more residual quantization errors can be captured and enhance the decoded image quality. Furthermore, in this work we use the block size of  $1 \times 1$  in the lower subbands. The advantage is as explained as above. The block size is set to  $2 \times 2$  at levels one and two, and  $1 \times 1$  for levels three and four. Figure 7 shows the effect of “blowing out” technique and the results are compared to Man’s codec [2]. In this scheme the image is decomposed to four DWT levels, and tested on image “Lena” of size  $512 \times 512$ . The incremental results for image compression scheme with  $2 \times 2$  block size for all four levels of WT can be found in [23]. In addition, the performance of MLVQ at 0.17 bpp ( $>32$  dB) which is better as compared to the result found in [3] for image lena with PSNR 30.3 dB.

### 5.2. Comparison with other VQ coders

Besides comparison with Man’s LVQ [2], we also include the comparison with other VQ works that incorporate the concept of EZW zerotree prediction. Therefore, we compare the

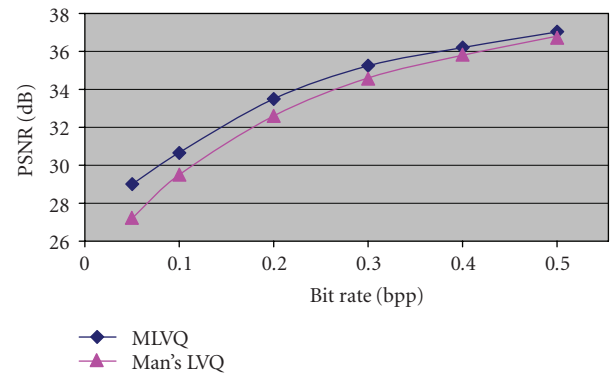
FIGURE 7: Comparison with Man’s LVQ [2] for image Lena  $512 \times 512$ .

TABLE 1: Performance comparison at bit rate 0.2 bpp.

Grey image	VLVQ-VSPHIT (entropy coded)	VSPECK	MLVQ	JPEG 2000
Lena	32.89	33.47	33.51	32.96
Goldhill	29.49	30.11	30.21	29.84
Barbara	26.81	27.46	27.34	27.17

MLVQ without adaptive threshold algorithm with the VLVQ of VSPHIT presented in [12]. In addition, the comparison is also made with the VSPECK image coder presented in [14]. Table 1 shows the comparison between the coders at 0.2 bpp for standard test images “Lena,” “Goldhill,” and “Barbara.” The comparison with JPEG 2000 is also included as reference so that the results in Section 5.3 on the effect of adaptive thresholding algorithm become meaningful. The table shows that MLVQ performs superior to VLVQ-VSPHIT for all three test images and better than VSPECK for test images “Lena” and “Goldhill.”



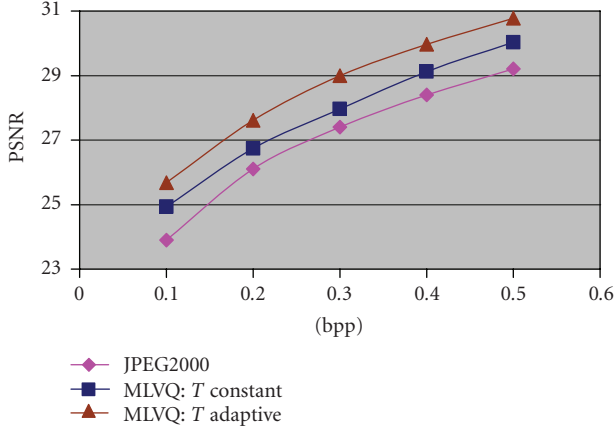


FIGURE 8: Test image "goldhill."

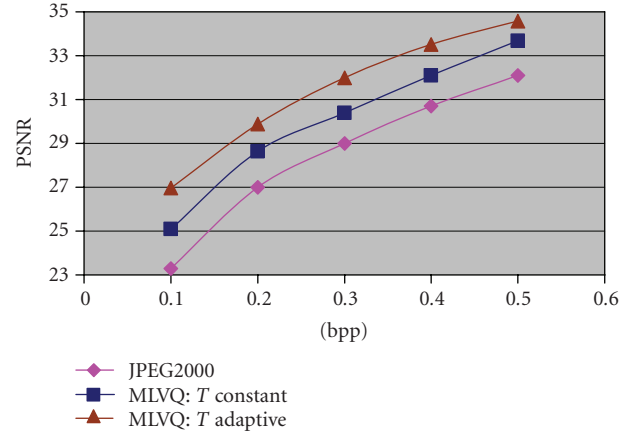


FIGURE 10: Test image "lena."

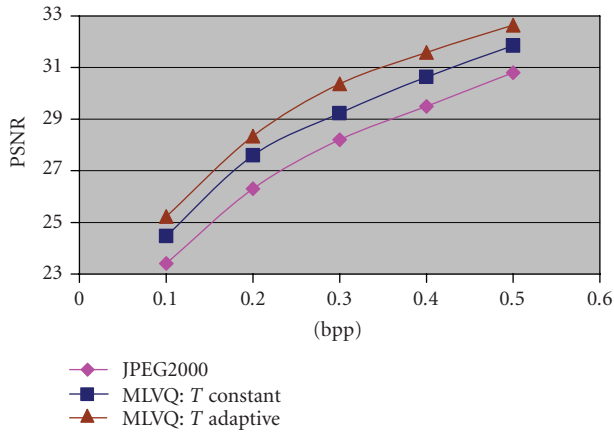


FIGURE 9: Test image "camera."

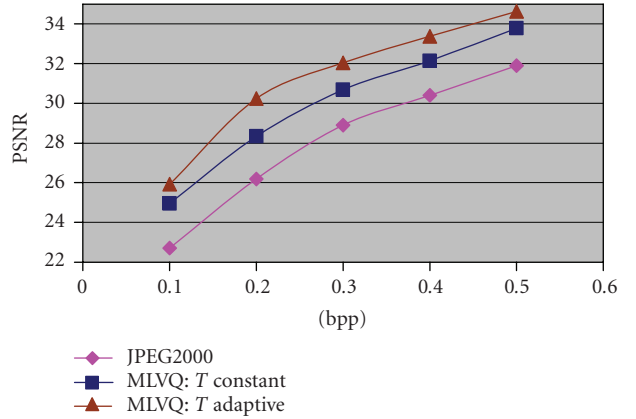


FIGURE 11: Test image "clown."

### 5.3. Effect of adaptive thresholding

The grey (8-bit) "Goldhill," "camera," "Lena," and "Clown" images of size  $256 \times 256$  are used to test the effect of adaptive subband thresholding to the MLVQ image compression scheme. The block size is set to  $2 \times 2$  at level one, and  $1 \times 1$  for levels two and three. The performance results of the new image coding scheme with constant and adaptive threshold are compared with JPEG 2000 [9], respectively, as shown in Figures 8, 9, 10, 11. It is clear that using the adaptive subband thresholding algorithm with MLVQ gives superior performance to either the JPEG 2000 or the constant subband thresholding with MLVQ scheme.

Figure 12 shows the visual comparison of test image "Camera" between the new MLVQ (adaptive threshold) and JPEG 2000 at 0.2 bpp. It can be seen that the new MLVQ (adaptive threshold) reconstructed images are less blurred than the JPEG 2000 reconstructed images. Furthermore it produces 2 dB better PSNR than JPEG 2000 for the "camera" test image.

TABLE 2: Computational complexity based on grey Lena of  $256 \times 256$  (8 bit) at bit rate 0.3 bpp.

Codec 1		Codec 2	
Total CPU time (s)	23.98	Total CPU time (s)	180.53
Constant threshold	0.0%	Adaptive threshold	86.3%
MLVQ	100%	MLVQ	13.7%

### 5.4. Complexity analysis

As the proposed algorithm is an iterative process, its computational complexity is higher when the adaptive thresholding algorithm is used (codec 2) as compared to the constant threshold (codec 1) as shown in Table 2. The threshold evaluation stage of the adaptive subband thresholding procedure illustrated in Figure 3(a) can be removed to reduce the computational cost with a resulting reduction in performance. In this evaluation, the Intel P4 (Northwood) with 3 GHz CPU clock speed, 800 MHz front side bus (FSB) and 512 MB RAM is used as the evaluating environment.

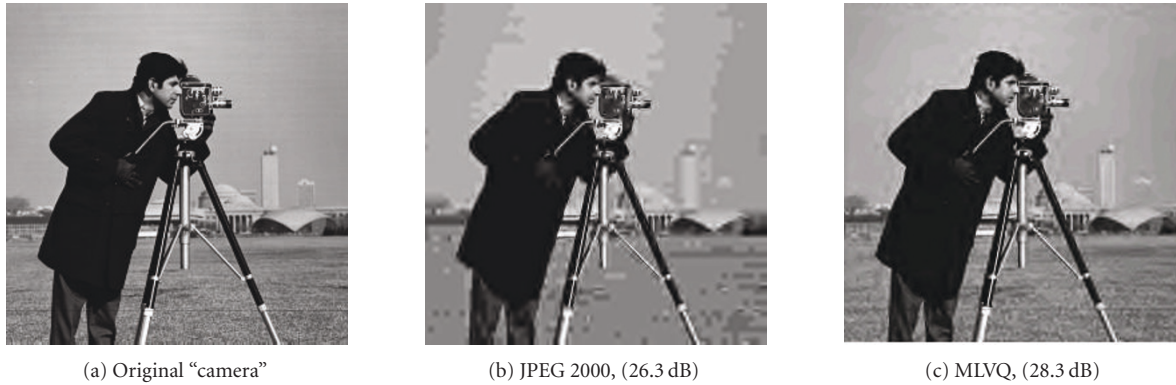


FIGURE 12

## 6. CONCLUSIONS

The new adaptive threshold increases the performance of the image codec which itself is restricted by the bit allocation constraint. The lattice VQ reduces complexity as well as computation load in codebook generation as compared to LBQ algorithm. This facilitates the use of multistage quantization in the coding scheme. The multistage LVQ technique presented in this paper refines the quantized vectors, and reduces the quantization errors. Thus the new multiscale multistage LVQ (MLVQ) using adaptive subband thresholding image compression scheme outperforms JPEG 2000 as well as other recent VQ techniques throughout all range of bit rates for the tested images.

## ACKNOWLEDGMENT

The authors are very grateful to the Universiti Sains Malaysia for funding the research through teaching fellowship scheme.

## REFERENCES

- [1] S. P. Voukelatos and J. Soraghan, "Very low bit-rate color video coding using adaptive subband vector quantization with dynamic bit allocation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 7, no. 2, pp. 424–428, 1997.
- [2] H. Man, F. Kossentini, and M. J. T. Smith, "A family of efficient and channel error resilient wavelet/subband image coders," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 9, no. 1, pp. 95–108, 1999.
- [3] M. Barlaud, P. Sole, T. Gaidon, M. Antonini, and P. Mathieu, "Pyramidal lattice vector quantization for multiscale image coding," *IEEE Transactions on Image Processing*, vol. 3, no. 4, pp. 367–381, 1994.
- [4] T. Sikora, "Trends and perspectives in image and video coding," *Proceedings of the IEEE*, vol. 93, no. 1, pp. 6–17, 2005.
- [5] A. S. Akbari and J. Soraghan, "Adaptive joint subband vector quantisation codec for handheld videophone applications," *Electronics Letters*, vol. 39, no. 14, pp. 1044–1046, 2003.
- [6] D. G. Jeong and J. D. Gibson, "Lattice vector quantization for image coding," in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '89)*, vol. 3, pp. 1743–1746, Glasgow, UK, May 1989.
- [7] J. H. Conway and N. J. A. Sloane, *Sphere-Packings, Lattices, and Groups*, Springer, New York, NY, USA, 1988.
- [8] F. F. Kossentini, M. J. T. Smith, and C. F. Barnes, "Necessary conditions for the optimality of variable-rate residual vector quantizers," *IEEE Transactions on Information Theory*, vol. 41, no. 6, part 2, pp. 1903–1914, 1995.
- [9] A. N. Skodras, C. A. Christopoulos, and T. Ebrahimi, "The JPEG 2000 still image compression standard," *IEEE Signal Processing Magazine*, vol. 18, no. 5, pp. 36–58, 2001.
- [10] J. M. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," *IEEE Transactions on Signal Processing*, vol. 41, no. 12, pp. 3445–3462, 1993.
- [11] A. Said and W. A. Pearlman, "A new, fast, and efficient image codec based on set partitioning in hierarchical trees," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 6, no. 3, pp. 243–250, 1996.
- [12] D. Mukherjee and S. K. Mitra, "Successive refinement lattice vector quantization," *IEEE Transactions on Image Processing*, vol. 11, no. 12, pp. 1337–1348, 2002.
- [13] W. A. Pearlman, A. Islam, N. Nagaraj, and A. Said, "Efficient, low-complexity image coding with a set-partitioning embedded block coder," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 14, no. 11, pp. 1219–1235, 2004.
- [14] C. C. Chao and R. M. Gray, "Image compression with a vector speck algorithm," in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '06)*, vol. 2, pp. 445–448, Toulouse, France, May 2006.
- [15] A. O. Zaid, C. Olivier, and F. Marmouton, "Wavelet image coding with adaptive dead-zone selection: application to JPEG2000," in *Proceedings of IEEE International Conference on Image Processing (ICIP '02)*, vol. 3, pp. 253–256, Rochester, NY, USA, June 2002.
- [16] A. Chandra and K. Chakrabarty, "System-on-a-chip test-data compression and decompression architectures based on Golomb codes," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 3, pp. 355–368, 2001.
- [17] S. W. Golomb, "Run-length encodings," *IEEE Transactions on Information Theory*, vol. 12, no. 3, pp. 399–401, 1966.
- [18] J. Senecal, M. Duchaineau, and K. I. Joy, "Length-limited variable-to-variable length codes for high-performance entropy coding," in *Proceedings of Data Compression Conference (DCC '04)*, pp. 389–398, Snowbird, Utah, USA, March 2004.

- [19] A. A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*, Kluwer Academic, New York, NY, USA, 1992.
- [20] J. D. Gibson and K. Sayood, "Lattice quantization," in *Advances in Electronics and Electron Physics*, P. Hawkes, Ed., vol. 72, chapter 3, Academic Press, San Diego, Calif, USA, 1988.
- [21] N. J. A. Sloane, "Tables of sphere packings and spherical codes," *IEEE Transactions on Information Theory*, vol. 27, no. 3, pp. 327–338, 1981.
- [22] J. H. Conway and N. J. A. Sloane, "Fast quantizing and decoding algorithms for lattice quantizers and codes," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 227–232, 1982.
- [23] M. F. M. Salleh and J. Soraghan, "A new multistage lattice VQ (MLVQ) technique for image compression," in *European Signal Processing Conference (EUSIPCO '05)*, Antalya, Turkey, September 2005.

**M. F. M. Salleh** was born in Bagan Serai, Perak, Malaysia, in 1971. He received his B.S. degree in electrical engineering from Polytechnic University, Brooklyn, New York, US, in 1995. He was then a Software Engineer at Motorola Penang, Malaysia, in R&D Department until July 2001. He obtained his M.S. degree in communication engineering from UMIST, Manchester, UK, in 2002. He has completed his Ph.D. degree in image and video coding for mobile applications in June 2006 from the Institute for Communications and Signal Processing (ICSP), University of Strathclyde, Glasgow, UK.



**J. Soraghan** received the B.Eng. (first class honors) and the M.Eng.S. degrees in 1978 and 1982, respectively, both from University College Dublin, Dublin, Ireland, and the Ph.D. degree in electronic engineering from the University of Southampton, Southampton, UK, in 1989. From 1979 to 1980, he was with Westinghouse Electric Corporation, USA. In 1986, he joined the Department of Electronic and Electrical Engineering, University of Strathclyde, Glasgow, UK, as a Lecturer in the Signal Processing Division. He became a Senior Lecturer in 1999, a Reader in 2001, and a Professor in 2003. From 1989 to 1991, he was Manager of the Scottish Transputer Centre, and from 1991 to 1995, he was Manager of the DTI Centre for Parallel Signal Processing. Since 1996, he has been Manager of the Texas Instruments' DSP Elite Centre in the University. He currently holds the Texas Instruments Chair in Signal Processing in the Institute of Communications and Signal Processing (ICSP), University of Strathclyde. In December 2005, he became Head of the ICSP. His main research interests include advanced linear and nonlinear multimedia signal processing algorithms; wavelets and fuzzy systems with applications to telecommunications; biomedical; and remote sensing. He has supervised 23 Ph.D. students to graduation, holds three patents, and has published over 240 technical papers.

