2011

Variable Block Size Motion Compensation In The Redundant Wavelet Domain

Ahmed Abdelgadir Suliman
North Carolina Agricultural and Technical State University

VARIABLE BLOCK SIZE MOTION COMPENSATION IN THE REDUNDANT WAVELET DOMAIN

by

Ahmed Abdelgadir Suliman

A dissertation submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of
DOCTOR OF PHILOSOPHY

Department: Electrical Engineering
Major: Electrical Engineering
Major Professor: Dr. Robert Y. Li

North Carolina A&T State University
Greensboro, North Carolina
2011

ABSTRACT

Suliman, Ahmed Abdelgadir. VARIABLE BLOCK SIZE MOTION COMPENSATION IN THE REDUNDANT WAVELET DOMAIN. **(Major Advisor: Dr. Robert Li)**, North Carolina Agricultural and Technical State University.

Video is one of the most powerful forms of multimedia because of the extensive information it delivers. Video sequences are highly correlated both temporally and spatially, a fact which makes the compression of video possible. Modern video systems employ motion estimation and motion compensation (ME/MC) to de-correlate a video sequence temporally. ME/MC forms a prediction of the current frame using the frames which have been already encoded. Consequently, one needs to transmit the corresponding residual image instead of the original frame, as well as a set of motion vectors which describe the scene motion as observed at the encoder.

The redundant wavelet transform (RDWT) provides several advantages over the conventional wavelet transform (DWT). The RDWT overcomes the shift invariant problem in DWT. Moreover, RDWT retains all the phase information of wavelet coefficients and provides multiple prediction possibilities for ME/MC in wavelet domain. The general idea of variable size block motion compensation (VSBMC) technique is to partition a frame in such a way that regions with uniform translational motions are divided into larger blocks while those containing complicated motions into smaller blocks, leading to an adaptive distribution of motion vectors (MV) across the frame.

The research proposed new adaptive partitioning schemes and decision criteria in RDWT that utilize more effectively the motion content of a frame in terms of various block sizes. The research also proposed a selective subpixel accuracy algorithm for the motion vector using a multiband approach. The selective subpixel accuracy reduces the computations produced by the conventional subpixel algorithm while maintaining the same accuracy. In addition, the method of overlapped block motion compensation (OBMC) is used to reduce blocking artifacts. Finally, the research extends the applications of the proposed VSBMC to the 3D video sequences. The experimental results obtained here have shown that VSBMC in the RDWT domain can be a powerful tool for video compression.

School of Graduate Studies
North Carolina Agricultural and Technical State University

This is to certify that the Doctoral Dissertation of

Ahmed Abdelgadir Suliman

has met the dissertation requirements of
North Carolina Agricultural and Technical State University

Greensboro, North Carolina
2011

Approved by:

Dr. Robert Y. Li Major Professor	Dr. Jung H. Kim Committee Member
Dr. M. U. Bikdash Committee Member	Dr. Clinton B. Lee Committee Member
Dr. Kenneth Williams Committee Member	Dr. John Kelly Department Chairperson

Dr. Sanjiv Sarin
Dean of Graduate Studies

BIOGRAPHICAL SKETCH

Ahmed Abdelgadir Suliman was born on May 13, 1974, in Khartoum, Sudan. He received the Bachelor of Science degree in Telecommunication and Control Systems from Gazira University in 1998, an Associate degree in Telecommunication and Network Engineer Technology from the Guilford Technical Community College in 2002 and Master degree in Electrical Engineering from North Carolina A & T State University in 2007. He is a candidate for the PhD. degree in Electrical Engineering.

ACKNOWLEDGMENTS

I would like to express my most sincere gratitude to my advisor, Dr. Robert Y. Li, for all his great advice, guidance, patience, and his continuing support and encouragement which helped the completion of this dissertation. I am also grateful to my doctoral advisory committee members: Dr. Jung Kim, Dr. M. Bikdash, Dr. Clinton Lee, and Dr. Kenneth Williams for reviewing my dissertation and providing many corrections and helpful comments.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Video is one of the most powerful forms of multimedia because of the extensive information it delivers. Each video sequence contains substantial visual information, thereby requiring vast resources for storage and communication. Therefore, the compression of video sequences has been the focus of work by many researchers for several decades. Video sequences are highly correlated both temporally and spatially, a fact which makes the compression of video possible. Video compression exploits the temporal correlation, because the temporal interval between every two consecutive video frames is very small, and most likely the two frames will exhibit high similarity. To decorrelate a video sequence temporally, modern video coders employ motion estimation and motion compensation (ME/MC). ME/MC forms a prediction of the current frame using the frames which have been already encoded. Consequently, one needs to transmit the corresponding residual image instead of the original frame, as well as a set of motion vectors which describe the scene motion as observed at the encoder. Since the residual frame typically contains much less signal energy than the original frame and the motion vectors are relatively few, the total bit rate to encode the motion-estimated frame is usually much less than the total bit rate to encode each frame as a still image.

A number of motion-estimation algorithms (ME) have been developed in order to provide efficient prediction of scene motion between frames. ME schemes can generally

be categorized as either feature matching or region matching [1]. The most widely used region matching technique is the block matching method, in which the current frame is divided into small blocks. The previous frame, called the reference frame, is searched for the best matching block for a given block in the current frame, and the resulting motion vector, $(\Delta x, \Delta y)$ indicates the position of the best-matching block. To limit the computational complexity of the ME process, the search is usually limited to some window surrounding the block position in the reference frame. The procedure of block matching is illustrated in Figure 1.1 and the calculation of the residual frame is

$$\text{Diff}\ (x, y, t, \Delta t) = f(x, y, t) - f(x + \Delta x, y + \Delta y, t - \Delta t) \tag{1.1}$$

where $\text{Diff}\ (x, y, t, \Delta t)$ denotes the calculated residual image at a position (x, y) in a time period t-Δt, while $f(x, y, t)$ denotes the frame value at position (x, y) and time t. This block-based ME/MC approach to video coding was first introduced in [1].

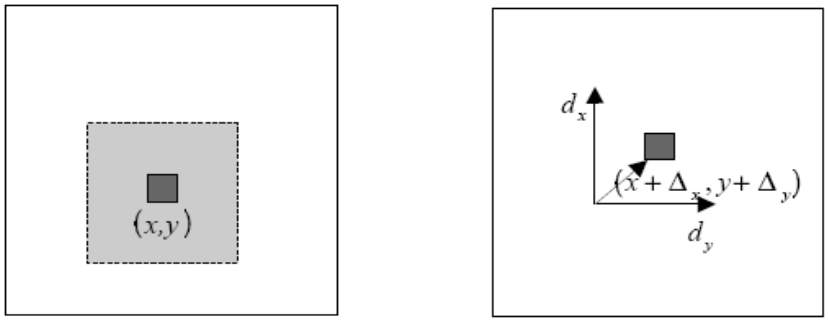


Figure 1.1 The block-matching algorithm. The dashed block shows the search window.

The block matching motion compensation can generally be categorized as either fixed block matching (FSBM) or variable size block matching (VSBM). The general idea of VSBM technique is to partition a frame in such a way that regions with uniform translational motions are divided into larger blocks while those containing complicated motions into smaller blocks, leading to an adaptive distribution of motion vectors (MV) across the frame. The VSBMC technique generally relies on a binary tree or a quadtree decomposition structure. Such a scheme is efficient in representing the partitioning, but the resulting blocks are restricted to be rectangular, and the sizes and locations of the blocks are also restricted by the tree structure.

Subpixel motion estimation plays an important role in compression efficiency within modern video codecs such as H.263 [2], [3] and MPEG-4 [4]. Subpixel motion estimation is implemented within these standards using interpolated values at 1/2 or 1/4 subpixel accuracy. Such interpolation gives a good reduction in residual energy for each predicted macroblock and therefore, improves compression. However, this leads to a significant increase in computational complexity at the encoder.

The research proposed a new adaptive partitioning scheme and decision criterion in the redundant wavelet domain that utilizes more effectively the motion content of a frame in terms of various block sizes. The proposed VSBMC deploys in two steps; splitting and merging. The redundant wavelet transform (RDWT) provides several advantages over the conventional wavelet transform (DWT). The RDWT overcomes the shift invariant problem in DWT. Moreover, RDWT retains all the phase information of wavelet coefficients and provides multiple prediction possibilities for ME/MC in wavelet

domain. As refinement for the block matching system, the research proposed a selective subpixel refinement algorithm for the motion vector using a multiband decision. The selective subpixel refinement reduces the computations produced by the conventional subpixel algorithm while maintaining the same accuracy.

In addition, the research extends the applications of the proposed VSBMC to the 3D video sequences. The 3D technology has been one of the fastest growing technologies in the recent years. Our approach is based on ME/MC techniques and the usage of depth-based rendering technique to reconstruct the desired stereoscopic views for each video frame. The depth image has a low energy and does not have sharp boundaries; therefore, it is not an easy task to obtain an accurate motion vector. Fortunately, the redundant wavelet domain provides a good solution by retaining all the phase information and provides a multiple prediction possibilities for motion techniques. Typically, a depth map is estimated from two images by calculating the parallax motion of pixels between the views. Consequently, a combination of only one texture and one depth video sequence is sufficient to provide appropriate rendering quality.

CHAPTER 2

REDUNDANT DISCRETE WAVELET TRANSFORM

2.1 Introduction

The main drawback of the DWT in the video compression is the shift variant that generates high frequency blocking artifacts which have big impact on the quality of ME/MC process when deployed in wavelet domain. To demonstrate the difficulty that the shift variance of the DWT poses in the task of tracking motion, consider the example illustrated in Figures 2.1 and 2.2. Shown in Figure 2.1 is a signal $s(n)$ and a shifted version of the signal $s(n-1)$. When Daubechies-Feauveau 9-7 filter is used to perform a 1-scale DWT on both $s(n)$ and $s(n-1)$, the effect of the shift variant, and the motion of the signal waveform is easily determined by comparing $s(n-1)$ to $s(n)$. However, in the wavelet domain, the low-band and high-band signals suffer from the shift-variant characteristic of the DWT [12]. In any event, the obtaining of accurate motion vectors for ME will not be possible using either the low-band or high-band signals in the DWT domain.

In order to overcome the shift variance of DWT, a number of proposals [5–10] have been made to use an overcomplete, or redundant, wavelet transform for ME/MC since such a redundant discrete wavelet transform (RDWT) lacks subsampling and is thus shift invariant.

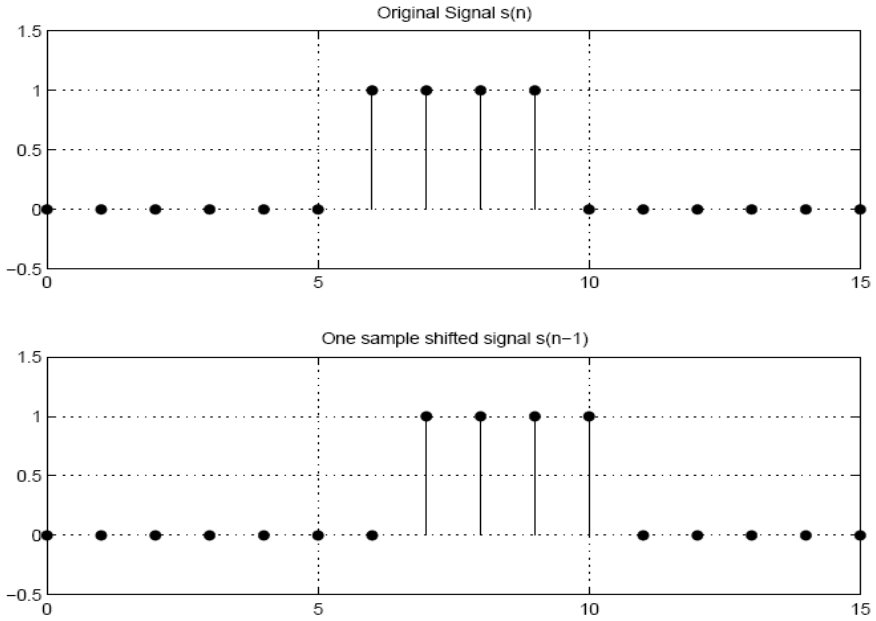


Figure 2.1 Signal $s(n)$ and its shifted version $s(n-1)$.

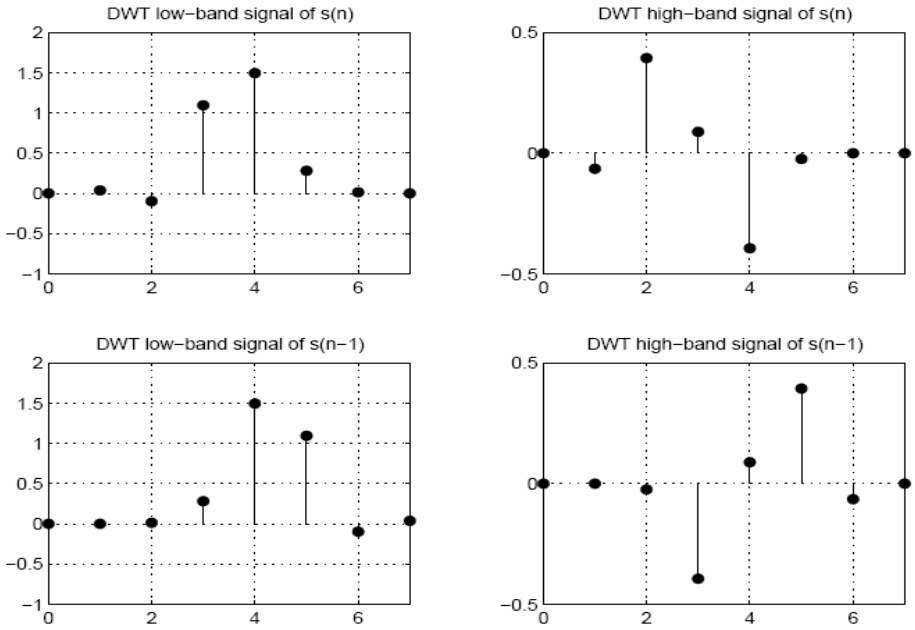


Figure 2.2 Wavelet-domain representation of $s(n)$ and $s(n-1)$.

2.2 RDWT versus DWT

The RDWT can be considered to be an approximation to the continuous wavelet transform that removes the downsampling operation from the conventional critically sampled DWT to produce an overcomplete representation [11]. The shift-variance characteristic of the DWT arises from its use of downsampling; while the RDWT is shift invariant since the spatial sampling rate is fixed across scale. To depict the implementation of the RDWT in terms of filter-banks, let us first illustrate the same for the DWT. A 1D DWT and its inverse are illustrated in Figure. 2.3. Consider $f[n]$ is the 1D input signal and $f'[n]$ is the reconstructed signal. $h[-k]$ and $g[-k]$ are the lowpass and highpass analysis filters, while the corresponding lowpass and highpass synthesis filters are $h[k]$ and $g[k]$. c_j and d_j are the lowband and highband output coefficients at level j. DWT analysis, or decomposition, is, mathematically [11],

$$c_j[k] = (c_{j+1}[k] * h[-k]) \downarrow 2 \qquad \text{and} \qquad d_j[k] = (c_{j+1}[k] * g[-k]) \downarrow 2 \tag{2.1}$$

where $*$ denotes convolution, and $\downarrow 2$ denotes downsampling by a factor of two. That is, if $y[n] = x[n] \downarrow 2$, then $y[n] = x[2n]$.

The corresponding operation of DWT synthesis, or reconstruction, is

$$c_{j+1}[k] = (c_j[k] \uparrow 2) * h[k] + (d_j[k] \uparrow 2) * g[k] \tag{2.2}$$

where $\uparrow 2$ denotes upsampling by a factor of two. That is, if $y[n] = x[n] \uparrow 2$, then

$$y[n] = \begin{cases} x[n/2], & n \text{ even} \\ 0, & n \text{ odd} \end{cases} \tag{2.3}$$

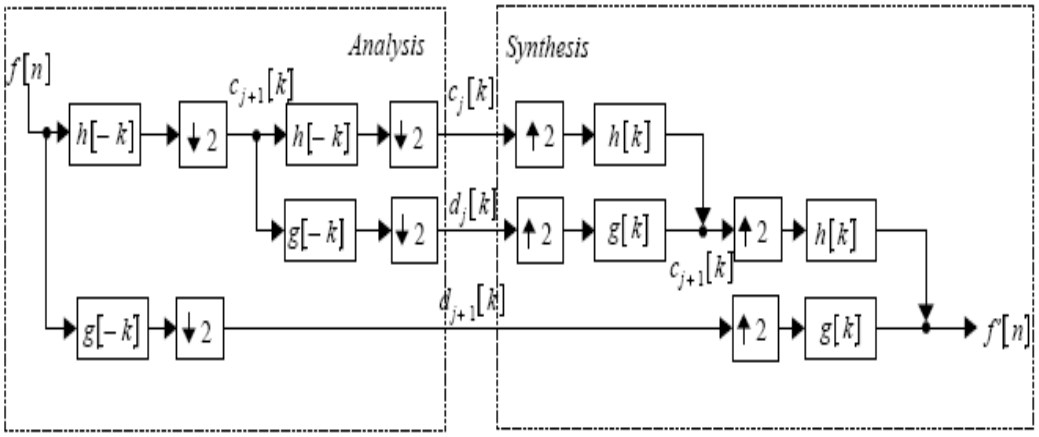


Figure 2.3 Two level 1-D DWT analysis and synthesis filter banks.

In contrast, a 1D- RDWT and its inverse are illustrated in Figure 2.4. The RDWT eliminates downsampling and upsampling of coefficients, and at each scale, the number of output coefficients doubles that of the input. The filters themselves are upsampled to fit the growing date length [12]. Specifically, the filters for scale j are:

$$h_j[k] = h_{j+1}[k] \uparrow 2 \qquad \text{and} \qquad g_j[k] = g_{j+1}[k] \uparrow 2 \tag{2.4}$$

RDWT analysis is then

$$c_j[k] = (c_{j+1}[k] * h_j[-k]) \quad \text{and} \quad d_j[k] = (c_{j+1}[k] * g_j[-k]) \tag{2.5}$$

While the RDWT synthesis is

$$c_{j+1}[k] = \frac{1}{2}(c_j[k] * h_j[k] + d_j[k] * g_j[k]) \tag{2.6}$$

Equations (2.4) through (2.6) are known as the *algorithme `a trous* [13], since the filter-upsampling procedure inserts "holes" ("trous" in French) between the filter taps.

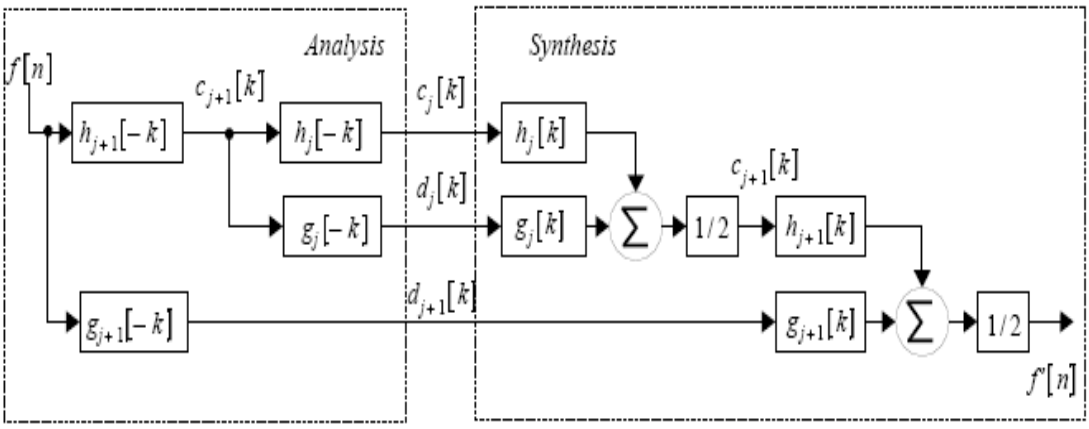


Figure 2.4 Two level 1-D RDWT analysis and synthesis filter banks.

2.3 RDWT Implementation and Coefficient Representation

There are several ways to implement the RDWT, and several ways to represent the resulting overcomplete set of coefficients. The most obvious implementation is a direct implementation of the *algorithme `a trous*, and results in subbands that are exactly the same size as the original signal, as is illustrated for a 1D signal in Figure 2.5. The advantage of this "spatially coherent" representation is that each RDWT coefficient is

located within its subband in its spatially correct position. Through appropriately subsampling each subband of an RDWT, one can produce exactly the same coefficients as does a critically sampled DWT applied to the same input signal. In fact, in a j-scale 1D- RDWT, there exist 2^j distinct critically sampled DWTs corresponding to the choice between even- and odd-phase subsampling at each scale of decomposition [14].

The most popular coefficient-representation scheme employed in RDWT-based video coders is that of a "coefficient tree," as illustrated in Figure 2.6 for a 1D signal. This tree representation is easily created by employing filtering and downsampling as in the usual critically sampled DWT; however, all "phases" of downsampled coefficients are retained and arranged as "children" of the signal that was decomposed. The process is repeated on the lowpass bands of all nodes to achieve multiple decomposition scales. Figure 2.6 shows an approximation and detail coefficients at scale J, as L_j and H_j, respectively. *E* indicates even-phase subsampling; *O* indicates odd-phase subsampling [14]. A path from root to leaf indicates a distinct critically sampled DWT; a j-scale RDWT consists of 2^j such DWTs. It is straightforward to see that each path from root to leaf in the RDWT tree constitutes a distinct critically sampled DWT, and there are 2^j such critically sampled DWTs in a j-scale decomposition [15].

An alternative, and equivalent, implementation of the RDWT tree representation comes from employing consistent subsampling phase and shifting the lowpass bands by one sample to generate children in the tree.

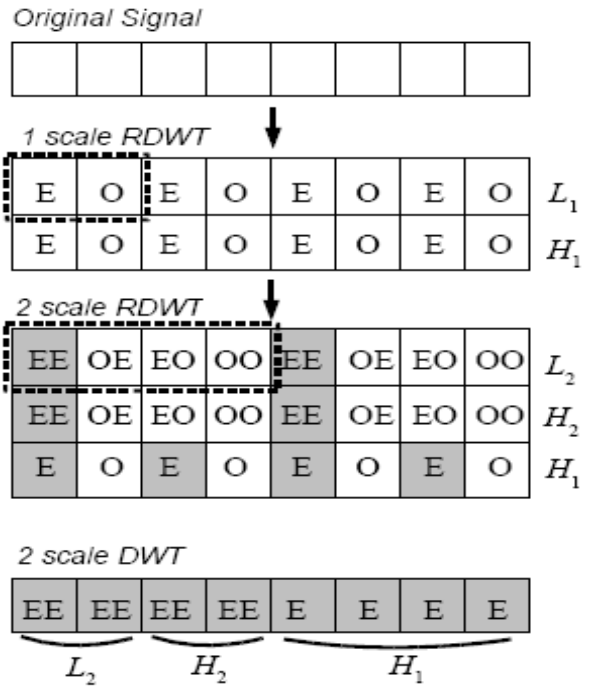


Figure 2.5 Spatially coherent representation of a two-scale 1D- RDWT.

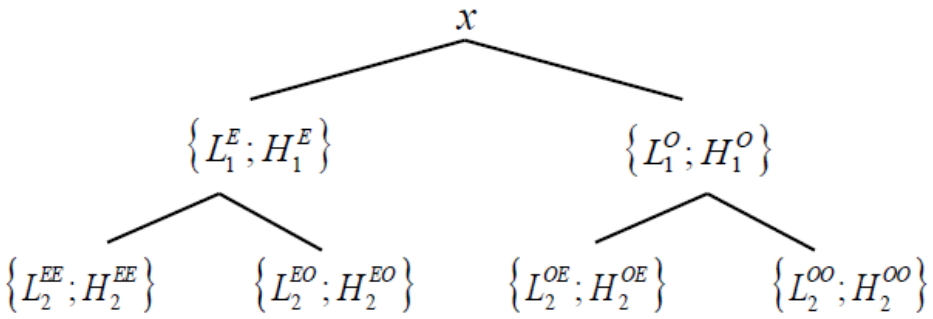


Figure 2.6 Tree representation of a two-scale RDWT of 1D-signal x .

The situation is similar for 2D decompositions implemented with separable 1D transforms, as illustrated in Figure 2.7. Figure 2.7 shows a j-scale 2D RDWT consisting of 4^j distinct critically sampled DWTs. The spatially coherent representation

of this two-scale 2D-RDWT means that the wavelet coefficients retain their correct spatial location within each subband, and each subband is the same size as the original image. In Figure 2.8 the notations B_j, H_j, V_j and D_j, denote the baseband, horizontal, vertical, and diagonal subbands, respectively, at scale j. This figure shows an example of RDWT process applied to the first frame of "Susie" sequence.

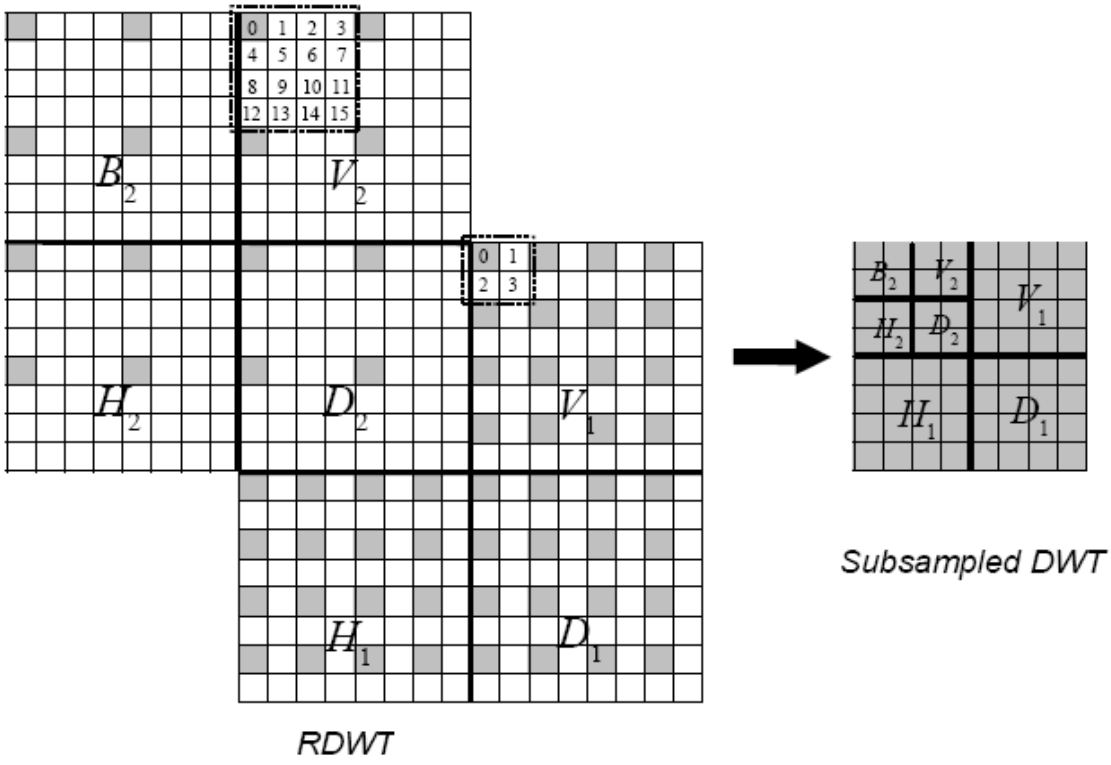


Figure 2.7 Spatially coherent representation of a two-scale 2D-RDWT.

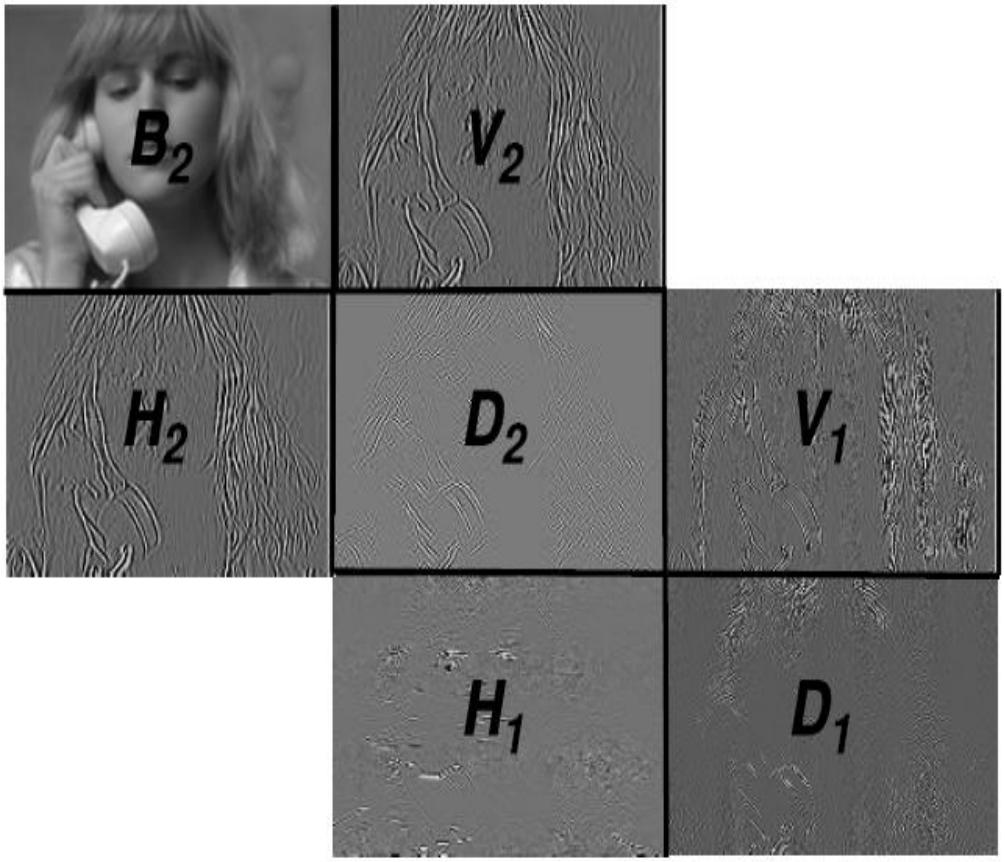


Figure 2.8 An example of a two-scale 2D-RDWT.

CHAPTER 3

BLOCK MATCHING ALGORITHM

3.1 Introduction

Motion estimation is a type of video compression scheme. The motion estimation process is designed to find the motion vector pointing to the best prediction macroblock in a reference frame. Compression redundancy between adjacent frames can be exploited whenever a frame is selected as a reference and subsequent frames are predicted from the reference using motion estimation. Block-based matching algorithms are the most popular methods for motion estimation and have been applied to most of video applications.

3.2 Block Matching Motion Estimation

In block-matching, a frame is divided into an array of macroblocks (MBs) [16]. Each MB has the size of $N \times N$ and is then compared with the candidate blocks in the reference frame. The candidate MB that is selected is the one that matches closest to the current block. Typically, two measurements, mean of absolute differences (MAD) and sum of squared differences (SSD) are adopted to evaluate how closely a candidate MB matches the current one [17]. Some video compression standards limit the maximum number of bits to encode each motion vector, thus restricting a motion vector's magnitude and its horizontal and vertical components' maximum value. In such case, the maximum value of the distance between a macroblock and its candidate reference blocks

is also limited. Usually, motion estimation is carried out only within a region of the reference frame, which is called the "search area". This also reduces the amount of computation for motion estimation.

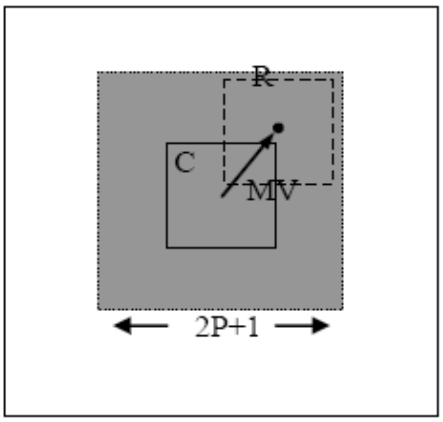
The search for the best matching MB is confined to a search area whose size is decided by the search parameter p. The search range is up to p pixels on all four sides of the corresponding MB in the reference frame. Figure 3.1 demonstrates a block-matching with search parameter p = P. The square in gray is the search area for block-matching. Usually, faster motions require a larger p value. The larger the search parameter, the more computationally intensive the process of motion estimation becomes.

$$\mathrm{MAD} = \frac{1}{N^2} \sum_{(i,j) \in current} | C_{ij}^{block} - R_{(i+u, j+v)} | \tag{3.1}$$

$$\mathrm{SSD} = \sum_{(i,j) \in current} (C_{ij}^{block} - R_{(i+u, j+v)})^2 \tag{3.2}$$

where C_{ij} and $R_{(i+u, j+v)}$ are the pixels being compared in the current MB and the MB on the reference frame, respectively. N is the size of the MB.

The most direct way to perform motion estimation is to exhaustively check every possible candidate MB within the search area on the reference frame, and chose the best matching one. This method is called full search block-matching algorithm (FSBMA) [20]. After block-matching, a motion vector (MV) is obtained for each MB.



C: Current block R: Block in the reference frame

Figure 3.1 Block-matching with search parameter p = P.

The motion vector is the displacements from the location of the current MB to the location of the best matching MB on the reference frame. Different coding techniques are usually used here to encode the MVs and generate bits for the video bit stream. MVs are used in motion compensation to construct the motion compensated frames. The difference between the current MB and the best-matching block is the prediction error which is usually encoded using the techniques that are used for compressing still-images. Notice that the reference frame is not necessarily the frame displayed before the current frame. Sometime, multiple reference frames are used. For example, if two reference frames are used: one frame before the current frame and one frame after the current frame in the display order but encoded previously. Thus the block matching is implemented on both reference frames, and the best matching-block is the one that has the least error among the candidate blocks on both reference frames.

3.3 Three-Step Search

Three-step search (TSS) [19, 20] is a fast searching algorithm to find the MVs. TSS consists of three steps, each step uses a fixed search pattern of nine uniformly spaced search points. In the first step, the point giving the least error is chosen and becomes the new search center for the next step search. The size of the search pattern is reduced by half from one step to the other, and the search points get closer after every step. The algorithm halts in three steps. TSS requires a fixed (9+8+8) = 25 search points for each block chosen as shown in Figure 3.2.

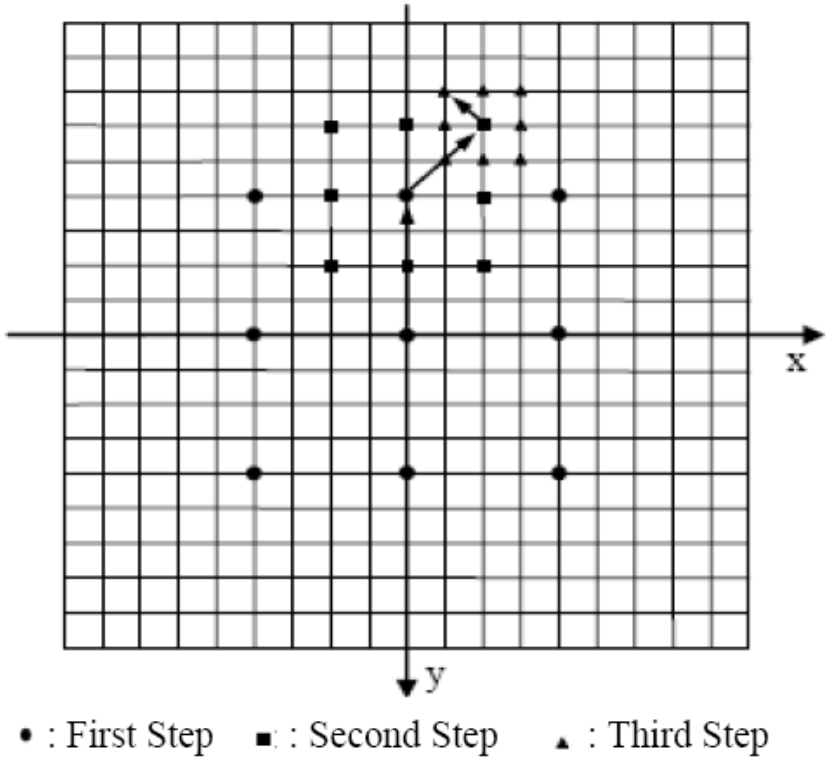


Figure 3.2 Three-step search process.

3.4 Group of Pictures

If a frame is decoded with error, all the frames that use it as the reference frame will be affected and decoded wrongly, thus the error propagates. To avoid such problem, one kind of video frame "I" frame is used. This type of frame doesn't use reference frames for encoding and is encoded by itself as a still-image. In the case when a frame is decoded not correctly, the error propagation will stop at the next I frame and the frames after that I frame in the encoding order will not be affected. Besides I frames, there are other two types of frames, "P frames" and "B frames". P frames use only a previously displayed frame as the reference frame [18, 21]. B frames use frames both in future and previous position in the display order as the reference frames. Figure 3.3 gives an example sequence of video frames.

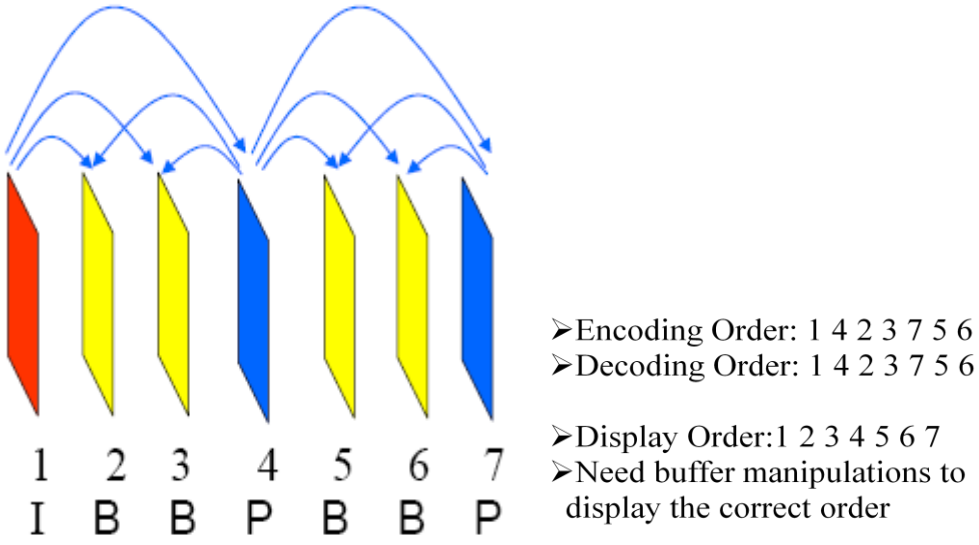


Figure 3.3 An example sequence of MPEG frames and the inter-frame dependencies.

3.5 Block-Based Motion Compensation

When decoding a video, motion compensation is carried out. The process uses the reference frames and the motion vectors to reconstruct each MB of the current frame. For motion vectors having integer components, the predicted MB is a simple copy and paste of the matching-block in the reference frame. For motion vectors having non-integer components, interpolation is used to estimate the MB for non-integer locations. After obtaining the prediction of each MB, the prediction of the whole frame is also obtained. The prediction error is then decoded and added to the frame, and the final motion compensated frame is reconstructed. To evaluate the quality of a reconstructed image, a popular metric is mean-squared-error (MSE) [21], which is the sum of the squared error between the motion compensated image and the original one as given by

$$MSE = \frac{1}{MN}\sum_{y=1}^{M}\sum_{x=1}^{N}(I(x,y) - \tilde{I}(x,y))^2 \tag{3.3}$$

here N and M are the number of rows and columns of pixels of the frame, respectively. $I(x,y)$ and $\tilde{I}(x,y)$ are the values of the intensity of a pixel at the position (x,y) in the original image and motion compensated picture, respectively. Another widely used metric for comparing various image compression techniques is the peak-signal-to-noise-ratio (PSNR). The measurement evaluates the image quality based on the root of MSE of the reconstructed frame. The mathematical formulae for PSNR is

$$PSNR = 10\log_{10}\left(\frac{(\text{peak to peak value of original data})^2}{\text{mean squared error}}\right) = 10\log_{10}.\left(\frac{I_{\max}^2}{MSE}\right) \tag{3.4}$$

where I_{max} is the maximum possible value of the pixels on the image. When 8 bits sample precision is used, the value of I_{max} is 255. The higher the value of PSNR, the better the quality of the compensated image.

Figure 3.4 illustrates the typical procedure of motion compensation. The computation requirement for motion compensation is much less than that of motion estimation. For each MB, motion estimation must calculate MAD or SSD on a number of $N \times N$ pixel blocks, whereas motion compensation just does the simple duplicate or interpolation of the selected matching block. This difference is critical and makes video decoding a much computationally simpler process than video encoding.

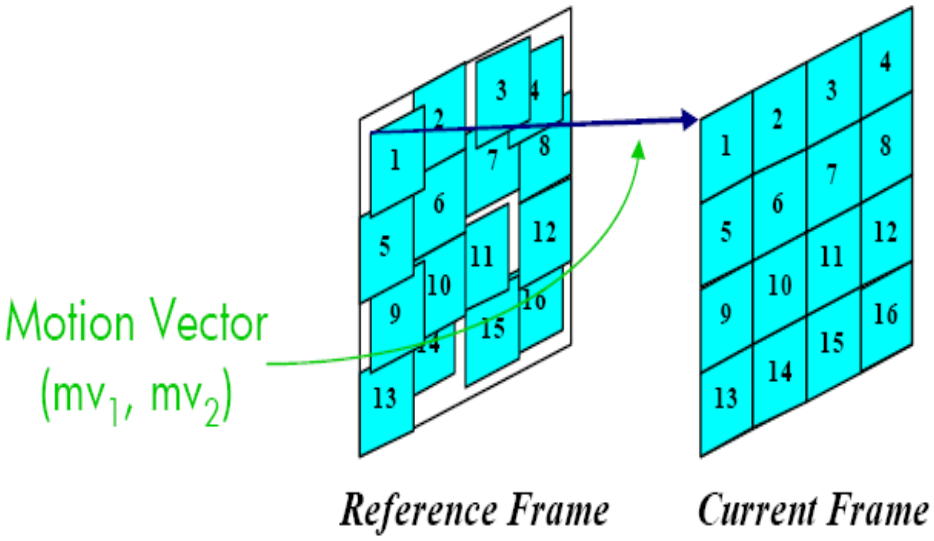


Figure 3.4 An illustration of motion compensation.

3.6 Variable Size Block Matching

In block matching motion compensation, there is a direct relationship between the size of the block and the error (or difference) between the current block and the best matching block in the reference frame [22]. As the block gets larger, the error is also likely to get larger because all the pixels in the block are unlikely to experience the same translational motion. Consequently, a single block size is then insufficient to control the error. An ideal VSBM technique should find the optimal tradeoff between the size of the blocks (and hence the number of blocks), and the total error associated with them.

VSBM algorithm imposes a complete quad-tree on the block structure of a frame. Let us denote a square block by (x, y, s) where (x, y) are the coordinates of the upper left-most pixel of the block, and s is the length of one side of the block. The frame is initially divided into identical-sized small blocks of size $s_{\min}$, they constitute the leaves of the tree [23]. We refer to the root of a tree as node E_x, and the four children of a node as $E_{4x}, E_{4x+1}, E_{4x+2}$ and E_{4x+3}. The output of block matching motion estimation is a set of non-overlapping blocks which together will cover the entire frame. This principle is illustrated in Figure 3.5 [23]. Clearly, there are many tree structures, and one can easily observe that any tree with height less than $\log 4n$, where n is the total number of blocks of size $s_{\min}$, can be mapped uniquely to a set of non-overlapping blocks which covers the entire input frame.

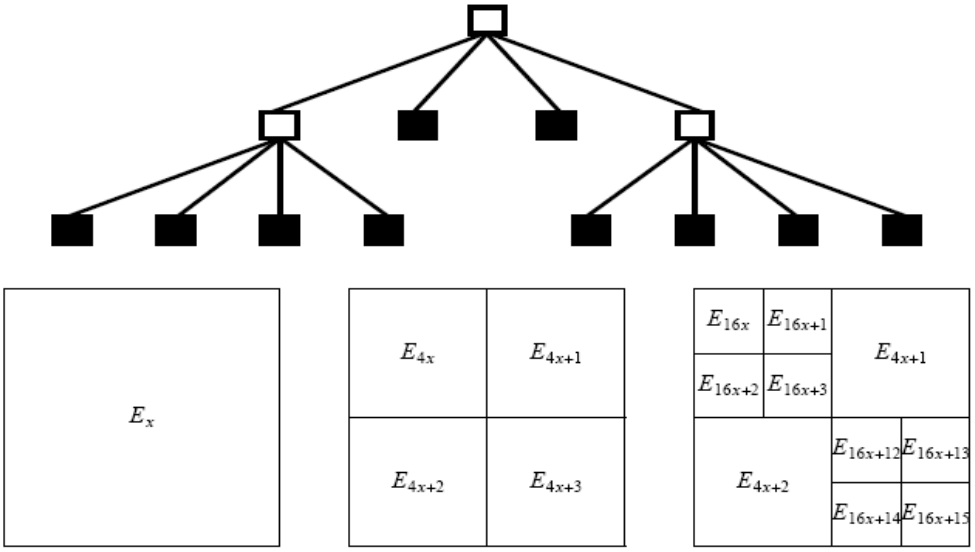


Figure 3.5 Decomposition and the resulting quad-tree.

The error of the tree (the total error of the matched blocks comprising the tree) is the error of the motion compensated frame. Given a required number of blocks *B* and two consecutive frames f_{i-1} and f_i, the block matching requirement is to find a tree with *B* leaves whose error is minimal among all possible trees with *B* leaves. Let $T_x(B)$ be the tree whose root is x and which covers only the area of the block corresponding to node *x* . Let $E_x(B)$ be the error of $T_x(B)$. Let π_B be the set of 4-tuple (*i, j, k, l*). Let $E_x(B)$ be the error of the block corresponding to node x. By solving this equation below, we can calculate the minimum error $E_0(B)$ and hence obtain the $T_0(B)$ for the entire frame.

$$E_x(B) = \min_{(i,j,k,l)\in\pi_B} \{E_{4x}(i) + E_{4x+1}(j) + E_{4x+2}(k) + E_{4x+3}(l)\} \tag{3.5}$$

3.7 Sub-pixel Motion Estimation

A key performance issue in motion estimation is sub-pixel accuracy. The theoretical and experimental analysis, such as the work done in [26], have established that sub-pixel accuracy has a significant impact on motion compensated prediction error performance for a wide range of natural moving scenes. As a consequence, recent efforts to standardize the compression methodology in video compression [2-4] have embraced the principle of sub-pixel accuracy for motion estimation and motion compensated prediction. The most popular techniques for subpixel image registration are based on interpolation. In this approach, the reference frame is bilinearly interpolated to obtain a new reference frame in sub-pixel accuracy. This half-pixel interpolation is illustrated Figure 3.6, where *A, B, C* and *D* indicate the integer pixels, while *a, b* and *c* are the interpolated half pixels. *a, b* and *c* are obtained by bilinear interpolation from *A, B, C* and *D* as,

$$a = (A + B)/2 \qquad b = (A + C)/2 \qquad c = (A + B + C + D)/4 \tag{3.6}$$

The block matching system is then modified so that the search is carried out with half-pixel accuracy in the interpolated reference frame. This incurs the addition of one more bit of precision to each component of the motion vectors. The concept of half-pixel accuracy can also be extended to quarter-pixel accuracy.

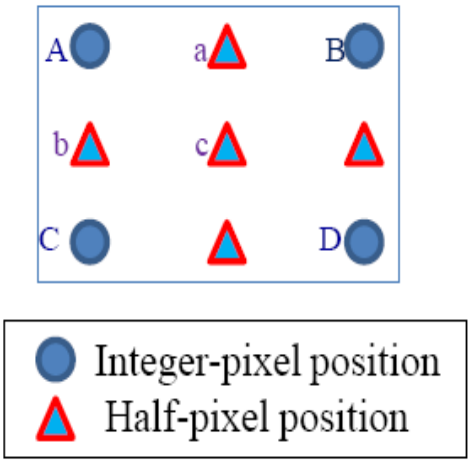


Figure 3.6 Half-pixel accuracy obtained by interpolation.

CHAPTER 4

NEW APPROACH OF MOTION ESTIMATION/MOTION COMPENSATION IN REDUNDANT WAVELET DOMAIN

4.1 Introduction

The research presents in this chapter a novel approach to VSBMC in the redundant wavelet domain, which incorporate the idea of multiband and VSBMC. The new approach recognizes the different phases in RDWT coefficients, and views the motion from different perspectives. This method allows partitioning the video frame more flexibly according to its motion content. The new adaptive partitioning scheme can utilize more efficiently the motion content of a frame in terms of the size and shape of the blocks developed. The partitioning information is efficiently represented by a two-bit coding scheme. The frame partitioning process is accomplished using two steps: first, splitting; second, merging.

As a refinement for the block matching system, the research proposed a selective algorithm for motion vector accuracy using a multiband-mode decision. The subpixel accuracy is a powerful tool to achieve more accurate coding, but it results in huge computational complexity since it uses a full search algorithm to find the accurate coordinate for each motion vector. The selective subpixel approach reduces the computations produced by the conventional subpixel algorithm while maintaining the same accuracy.

4.2 System Architecture for MB-VSBMC

The encoder of our multi-band VSBMC video-coding system (MB-VSBMC) is depicted in Figure 4.1. The current and reference frames are transformed into RDWT coefficients, and both ME and MC take place in the redundant wavelet domain. In a *J*-scale RDWT decomposition, each block in the original spatial domain corresponds to $3J + 1$ blocks of the same size, one for each subband. The collection of these co-located blocks is called a set. In the ME procedure, block matching algorithm is used to determine the MV of each set as a whole. Specifically, a block-matching procedure uses a cross-subband distortion measure that sums absolute differences for each block of the set. An adaptive variable size window is used for the block search. The all-phase correlation edge mask and approximation subband (*LL*) are used to construct a multiband decision criteria for choosing the block size.

After the block size is determined, the motion from the reference frame to the current frame is estimated in the RDWT domain, and motion vectors are transmitted to the decoder. Multiband MC is accomplished by using a multiple reference frames (subbands) algorithm to generate bidirectional prediction. Residing in the RDWT domain, the motion-compensated residual is itself redundant; consequently, it is down-sampled before coding. The final encoding step for coder/decoder (CODEC) consists of a set partitioning in hierarchical trees (SPIHT) algorithm for still image compression [27].

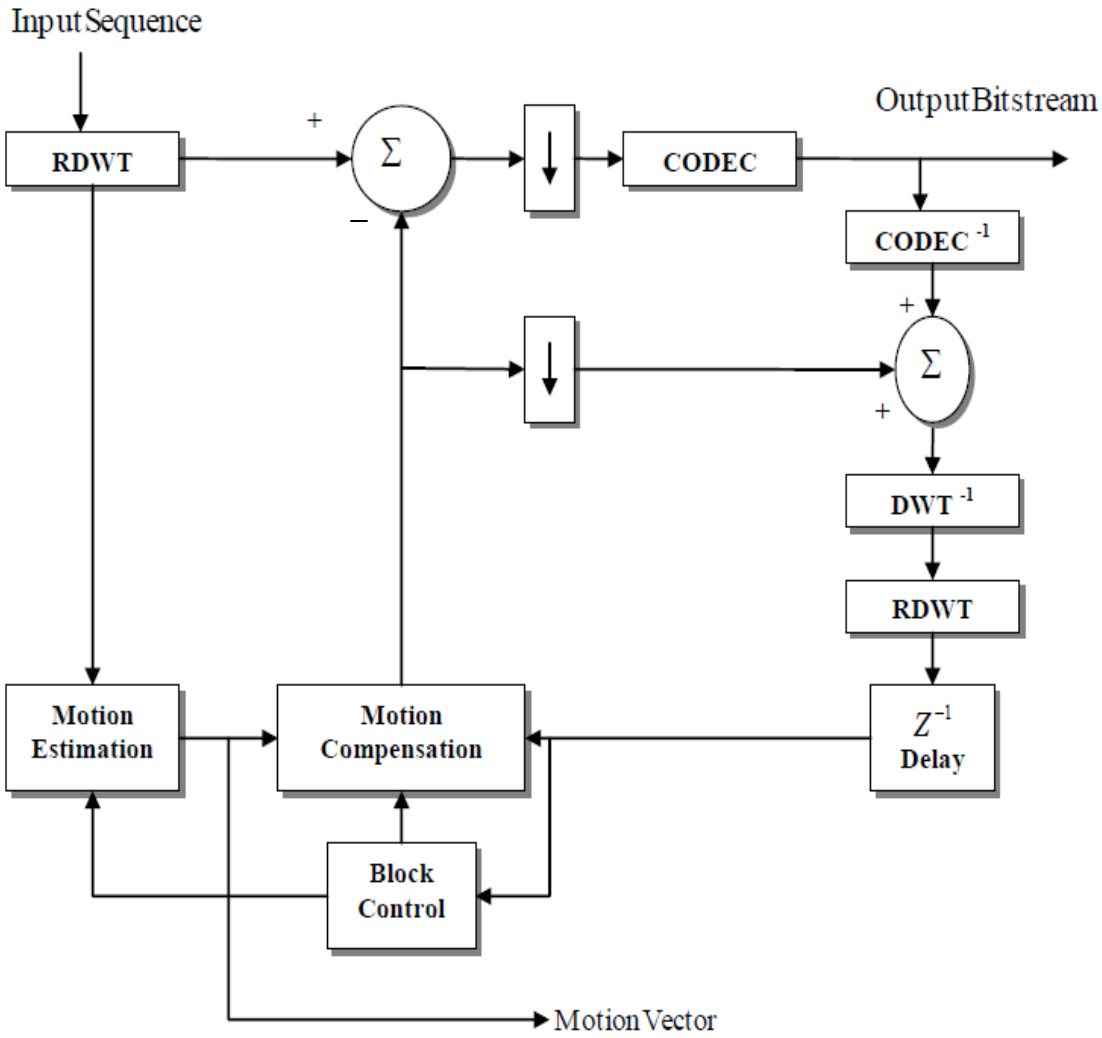


Figure 4.1 Block diagram of the MB-VSBMC video-coding system. CODEC uses the SPIHT algorithm.

4.3 Proposed Decision Criterion

The research proposed a new decision criterion that partitions a given frame into variable size regions according to the motion information of the frame. The partitioning information is efficiently represented by a two-bit coding scheme. The frame partitioning

is accomplished by: first, potentially splitting a 16×16 block into 8×8 blocks, and then 4×4 blocks; second, potentially merging four neighbors of 16×16 blocks into a 32×32 block.

4.3.1 Splitting Process

The general idea of the splitting is to divide a certain 16×16 MB into up to four sub-MBs of 8×8, then divide a certain 8×8 MB into up to four sub-MBs of 4×4. The research has developed five steps to accomplish that as shown in Figure 4.3:

First: For a given 32×32 MB, decide which 16×16 MB is a candidate to be split. As it was mentioned before, each frame has at least four subbands in redundant wavelet domain. The direct multiplication of the RDWT coefficients at adjacent scales (all-phase correlation edge mask) distinguishes important features from the background due to the fact that wavelet-coefficients are correlated across scales. We will use an all-phase correlation edge mask of the current frame to determine which 16×16 MB is a candidate to be split by setting a number of thresholds. The correlation edge mask acts as a map for the decision making of the variable block size, since it highlights the edges. To create the correlation edge mask for the frame, we multiply the vertical (V_j), horizontal (H_j), and diagonal (D_j) bands together across scales and combine the products; i.e.

$$\text{mask}(x,y) = \left|\prod_{j=J_0}^{J_1} V_j(x,y)\right| + \left|\prod_{j=J_0}^{J_1} H_j(x,y)\right| + \left|\prod_{j=J_0}^{J_1} D_j(x,y)\right| \qquad (4.1)$$

where J_0 and J_1 are the starting and ending scales, respectively, of the correlation operation. Note that mask(x, y) is the resulting correlation image with the same dimensions as the original image. See Figure 4.2 for an illustration of the correlation edge mask.

(a) The fourth frame of "News" sequence.

(b) Its corresponding correlation mask.

Figure 4.2 An illustration of the correlation edge mask.

Second: determine the global maximum of the mask,

$$\text{Mask}_{\max} = \max(\text{mask}(x, y)) \tag{4.2}$$

and set the threshold, τ as: $\tau = \alpha \cdot \text{Mask}_{\max}$ (4.3)

where the threshold parameter is $\alpha, 0 \le \alpha \le 1$.

Third: Divide the current correlation edge mask into 16×16 MBs and select each MB with an average value larger than the threshold for further splitting.

$$Block_{Avg}^{16\times16} \geq \tau \tag{4.4}$$

Fourth: For the chosen 16×16 MB from the last step, divide the current and the reference correlation mask into 8×8 MBs, and subtract the co-located 8×8 MBs from each other and then test the result against correlation threshold τ_1 .

$$Diff_{i,j} = \sum_{i,j}^{8} |\,\mathrm{mask}_{i,j}^{cur}\,| - |\,\mathrm{mask}_{i.j}^{ref}\,| \tag{4.5}$$

where $Diff_{i,j}$ is the absolute difference between two correlation masks. *i* and *j* are the horizontal and vertical displacements. $\mathrm{mask}_{i,j}^{cur}$ and $\mathrm{mask}_{i.j}^{ref}$ are co-located 8×8 MBs for the current and reference frame, respectively. Next, set the threshold τ_1 as:

$$\tau_1 = \frac{Diff_{i,j}^{\max} + Diff_{i,j}^{\min}}{2} \tag{4.6}$$

Then, select any 8×8 MB with its average mask value larger than the threshold for further splitting.

$$Block_{Avg}^{8\times8} \geq \tau_1 . \tag{4.7}$$

Fifth: Those selected 8×8 MBs from the last step can be split further into 4×4 MBs using the same procedure from above by changing the index from 8×8 to 4×4. The threshold becomes τ_2 . See Figure 4.3.

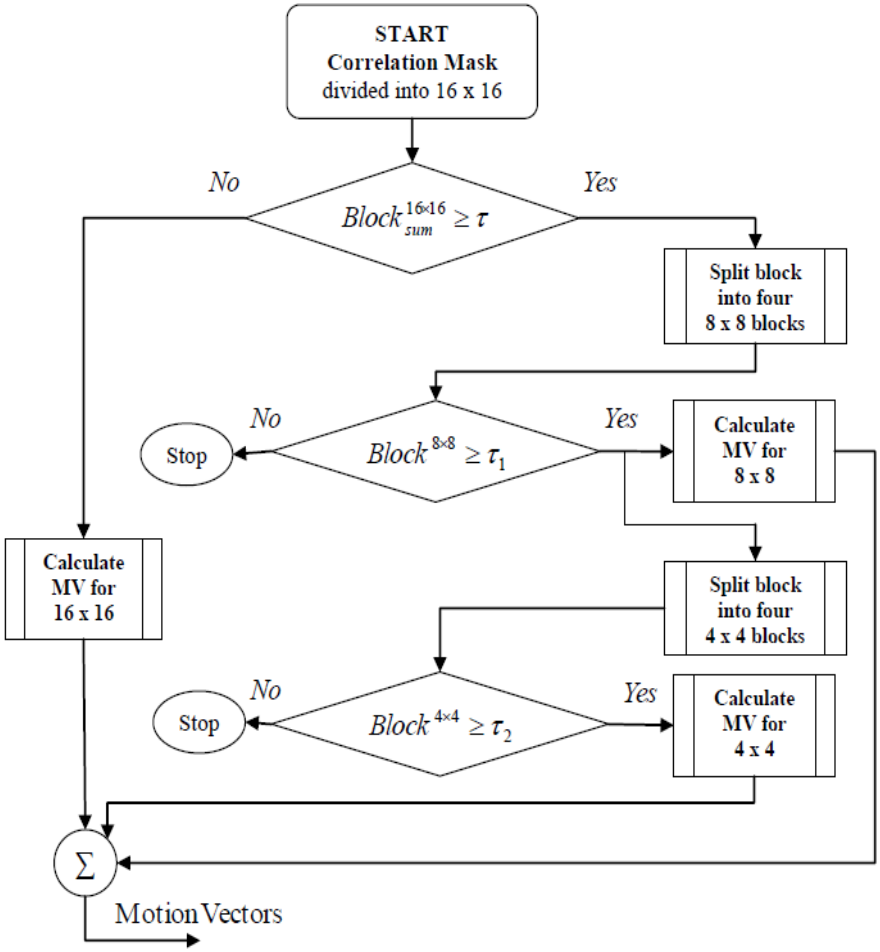


Figure 4.3 The splitting process.

4.3.2 Merging Process

The general idea of our merging process is to replace the potential four neighbors of 16×16 MBs that do not contain important motion content by a single 32×32 MB. The main purpose of the merging process is to reduce the unnecessary MVs by merging the MVs of (two, three or four) 16×16 MBs (little motion content) into one MV to represent them. To complete the merging process, start by dividing the approximation subband

(*LL*) of the reference and the current frame into 16×16 blocks, and subtract the co-located 16×16 blocks from each other and then test the result against the threshold τ_3.

$$DL_{i,j}^{LL} = \sum_{i,j}^{16} | LL_{i,j}^{cur} | - | LL_{i.j}^{ref} | \tag{4.8}$$

$$FD = \frac{DL_{i,j}^{\max} + DL_{i,j}^{\min}}{2} \tag{4.9}$$

set the threshold τ_3 as: $\tau_3 = \beta \cdot FD$, where the threshold parameter is β, $0 \le \beta \le 1$. The condition for merging is set to be: for each four neighbors of 16×16, if at least two of the four siblings (16×16) fall under the threshold, merge the siblings to a 32×32 block and calculate the MV for the 32 × 32 block.

4.4 VSBM Tree Construction

It is known that a quadtree data structure decomposes a $2^{l_{\max}} \times 2^{l_{\max}}$ image frame into an $(l_{\max} - l_0 + 1)$-level hierarchy, where a block at level l has a size of $2^l \times 2^l$, $0 \le l_0 \le l \le l_{\max}$. This structure corresponds to a tree, where each $2^l \times 2^l$ block (called a node) either can be a leaf (i.e., it cannot be further subdivided) or can be subdivided into four subblocks, each of size $2^{l-1} \times 2^{l-1}$. Thus, each subblock is a child node [49].

The tree can be represented by a bitstream where a “0” represents a leaf and a “1” represents a nonleaf node. To efficiently encode such a partitioning, a two-bit coding scheme is essential. In this scheme, each leaf or nonleaf node of the tree is represented by a two-bit code (TBC) [50]. The first bit is used to distinguish between a leaf and a

nonleaf node, while the second is used to indicate whether a motion vector is being transmitted. In the first-bit position of the code, a 0 or 1 represents, respectively, a leaf or a nonleaf node; in the second-bit position, a 1 represents the transmission of a motion vector and a 0 the lack of it. For example, the code "10" represents a nonleaf node with no motion vector being transmitted, while the code "01" represents a leaf with its motion vector being transmitted. It is noted that for a leaf with no motion vector, the decoder uses its nearest direct ancestor's motion vector as its own. Figure 4.4 shows an example of TBC applied to a 32×32 MB and its sub-MBs. In this example we will transmits three MVs. The MVs for the 16×16 and 8×8 sub-MBs are obtained from the splitting process. The other MV for the 32×32 MB is obtained from the merging process.

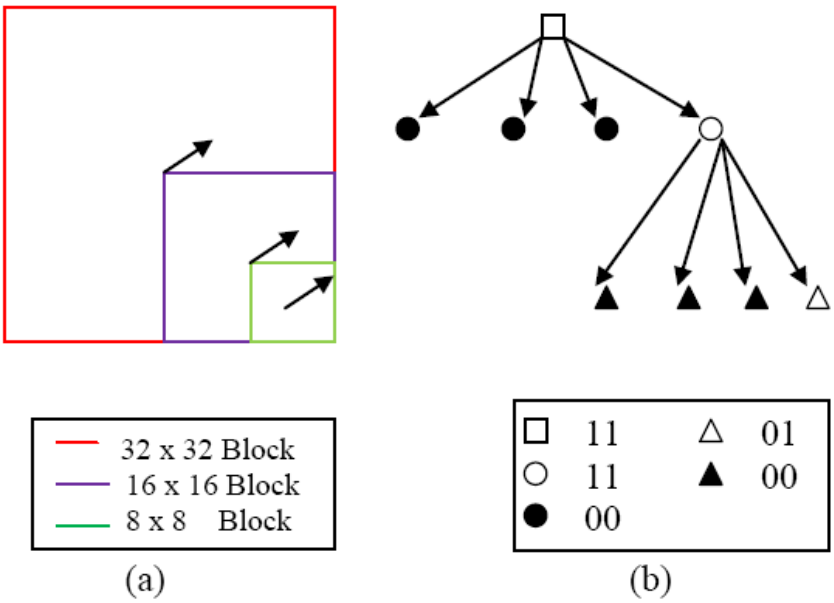


Figure 4.4 An example of the TBC applied to 32×32 MB and its sub-MBs.

4.5 Selective Refinement Algorithm

As a refinement for the block matching system, the research proposed a selective algorithm for motion vector accuracy to reduce its computational burden. The subpixel accuracy is a powerful tool to achieve high coding, but it results in huge computational complexity since it uses a full search algorithm to find the accurate coordinate for each motion vector [38]. To perform the subpixel motion estimation, the encoder interpolates pixel values at subpixel positions using pixel values at integer pixel positions in reference frames. Although the coding accuracy is highly increased by the subpixel motion estimation, the computational complexity of this repetitive subpixel motion search is very large in comparison with fast integer-pixel motion search. In other words, the subpixel motion estimation without considering the macroblock characteristics is not efficient in terms of the computational complexity. To reduce this additional complexity, a new method of selective refinement algorithm (Figure 4.5) is developed. The basic procedure works in the following two steps:

Step 1: Use the decision tree from the variable size block matching to decide the size of the block. Notice that we do not include a 32×32 block in this procedure, because we assume that most 32×32 blocks do not have detailed texture and most likely its motion vector is close to zero.

Step 2: Calculate the sum of absolute difference (SAD) for each 16×16 and 8×8 MBs in the correlation edge mask, and test them against a threshold λ_1 for 16×16 MB, and λ_2 for 8×8 MB. If a selected 16×16 MB has a SAD value less than λ_1, calculate half pixel accuracy; otherwise, calculate quarter pixel accuracy. If a selected 8×8 MB has a

SAD value less than λ_2, keep the integer accuracy unchanged; otherwise calculate half pixel accuracy.

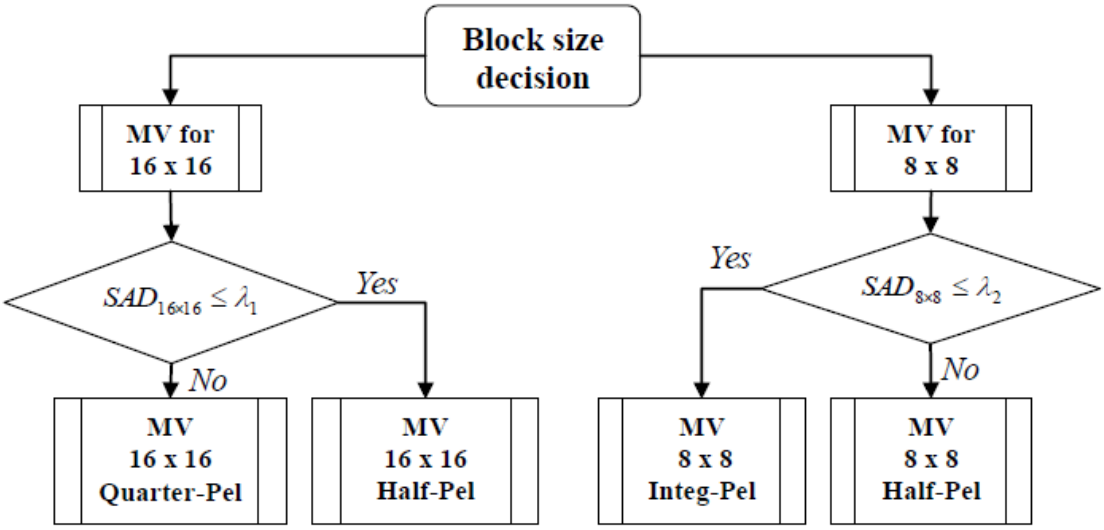


Figure 4.5 Selective refinement algorithm procedure.

The SAD is a sum of absolute difference between co-located MBs in the reference and current frame, and it can be calculated using the equation below.

$$SAD_{i,j} = \sum_{m=0}^{N}\sum_{n=0}^{N} | M_{curr(i,j)}(m,n) - M_{ref(i,j)}(m,n) | \qquad (4.10)$$

where $SAD_{i,j}$ is the sum of absolute difference at (i, j)-*th* MB, $M_{curr(i,j)}(m,n)$ is the current frame MB, and $M_{ref(i,j)}(m,n)$ is a co-located MB in the reference frame.

(m,n) is the pixel index within (i,j)-*th* MB, and *N* is 16 or 8 depending on the block size. We use the Equation 4.10 to calculate a combined SAD from the correlation edge mask and approximation band. To calculate the combined SAD, we calculate the SAD for each 16×16 MB in both the correlation edge mask and approximation band. Next, for every four neighbors of 16×16 MBs (32×32 block size), we pick a maximum and a minimum from these SAD values. Then, we plug these values in the equation below

$$SAD_{\lambda} = \frac{\sum_{i.j} SAD_{LL,Corr}^{\max} + \sum_{i.j} SAD_{LL,Corr}^{\min}}{4} \tag{4.11}$$

where $\sum SAD_{LL,Corr}^{\max}$ is the summation of the maximum SAD values from the correlation edge mask and approximation band; and $\sum SAD_{LL,Corr}^{\min}$ is the summation of the minimum SAD values from the correlation edge mask and approximation band. Finally, we set the thresholds λ_1 and λ_2 as: $\lambda_1 = \alpha_n \cdot SAD_{\lambda}$; and $\lambda_2 = 0.25 \cdot \alpha_n \cdot SAD_{\lambda}$. The threshold parameter is α_n, where $0 \le \alpha_n \le 1$.

4.6 Experimental Results

For the experiment, we use 60 frames of 352×288 "News" sequence, with common intermediate format CIF (standard video format used in videoconferencing); and 70 frames of 144×176 "Foreman" sequence, with quarter common intermediate format (QCIF). The sequences are grayscaled and have a temporal sampling of 25 frame/sec. The first frame is intra-encoded (I-frame) while all subsequent frames use ME/MC (P and

B-frames). All wavelet transforms (RDWT) use the Daubechies 9-7 filter with symmetric extension and a decomposition of $J = 2$ level. The parameters α, β and α_n are 0.4, 0.68 and 0.73, respectively. The core compression engine in all experiments is SPIHT. Since SPIHT produces an embedded coding, each frame of the sequence is coded at exactly the specified target rate with a compression rate of 0.5 bpp for I frame and 0.25 bpp for P and B frames. For comparison purposes, we use the peak signal-to-noise ratio (PSNR) and the structural similarity index (SSIM) [39]. The SSIM is a method for measuring the similarity between two images. It can be viewed as a quality measure of one of the images being compared, provided that the other image is regarded as of perfect quality [39].

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{4.12}$$

The mean for image *x* or *y* can be obtained using:

$$\mu_x = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{4.13}$$

The standard deviation for image *x* or *y* can be obtained using:

$$\sigma_x = \left(\frac{1}{N-1}\sum_{i=1}^{N} (x_i - \mu_x)^2 \right)^{1/2} \tag{4.14}$$

where N is the number of pixels, and the constant C_1 and C_2 are included to avoid instability when $(\mu_x^2 + \mu_y^2)$ is very close to zero. $C_1 = (K_1 L)^2$ and $C_2 = (K_2 L)^2$, where L is the dynamic range of the pixel values (255 for 8-bit grayscale images), and both $K_1 << 1$ and $K_2 << 1$are small constants.

As shown in Tables 4.1 – 4.4, the PSNR and SSIM averages of all frames were calculated for the coding system in both spatial and RDWT domains. In RDWT, the results include our proposed MB-VSBMC method, conventional FSBMC method (8×8 block size), and the conventional VSBMC wavelet method [40] by replacing the CODEC from DCT to SPIHT and applying the decision criteria to the wavelet approximation band. In addition subpixel accuracy and selective refinement algorithm are also included for comparison. For comparisons among FSBMC, conventional VSBMC, and MB-VSBMC, all without any sub-pixel accuracy in the RDWT; the proposed MB-VSBMC performed the best in terms of SSIM and PSNR. For comparisons among FSBMC, conventional VSBMC, and MB-VSBMC with sub-pixel accuracy in the RDWT; the proposed MB-VSBMC again performed the best in terms of SSIM and PSNR. For comparison between MB-VSBMC with sub-pixel accuracy and MB-VSBMC with selective sub-pixel accuracy, the selective approach has computational advantage without sacrificing much performance in terms of SSIM and PSNR.

Figure 4.6 shows the comparison of the compressed 4th frame for "News" sequence using three different block partitioning techniques in the redundant wavelet domain. Figure 4.6.a is the original 4th frame. Figure 4.6.b is the compressed frame using FSBMC (8×8 MBs) with subpixel accuracy. Figure 4.6.c is the compressed frame using

MB-VSBMC with subpixel accuracy. Figure 4.6.d is the compressed frame using MB-VSBMC with selective algorithm. Figure 4.7 also shows the comparison of a compressed 6^{th} frame for "Forman" sequence using the same three block partitioning techniques in the redundant wavelet domain as explained in the Figure 4.6.

Figure 4.8 shows an example of partitioning results using different approaches. Figure 4.8 (a) and (b) are the 4^{th} frame of the "News" sequence. Figure 4.8.a shows a MB-VSBMC partitioning using 16×16, 8×8 and 4×4 block sizes for the splitting process and 32×32 block size for merging process. The variation from the 32×32 to 4×4 block size will result in more accuracy by capturing the motion content. Figure 4.8.b shows a conventional VSBMC partitioning by starting to split from 32×32 down to 4×4 block size. Figure 4.8 (c) and (d) are the 6^{th} frame of the "Foreman" sequence. Figure 4.8.c shows a MB-VSBMC partitioning using 16×16 and 8×8 block sizes for the splitting process and 32×32 block size for merging process. Figure 4.8.d shows a conventional VSBMC partitioning by starting to split from 32×32 down to 8×8 block size.

Figure 4.9 shows the frame by frame comparison of PSNR for "News" sequence using scalable compression rate of 0.5 bpp for I and P, and 0.25 bpp for B frames. Figure 4.10 shows the frame by frame comparison of PSNR for "Foreman" sequence using scalable compression rate of 0.5 bpp for I and P, and 0.25 bpp for B frames. These two figures are related to the results in Table 4.4.

Table 4.1 Comparison between conventional VSBMC and FSBMC in spatial domain.

Spatial Domain	News		Forman	
	SSIM	PSNR	SSIM	PSNR
FSBMC	0.853	29.76	0.814	27.47
VSBMC	0.920	32.68	0.908	29.01

Table 4.2 Comparison between conventional VSBMC, FSBMC and MB-VSBMC without any sub-pixel accuracy.

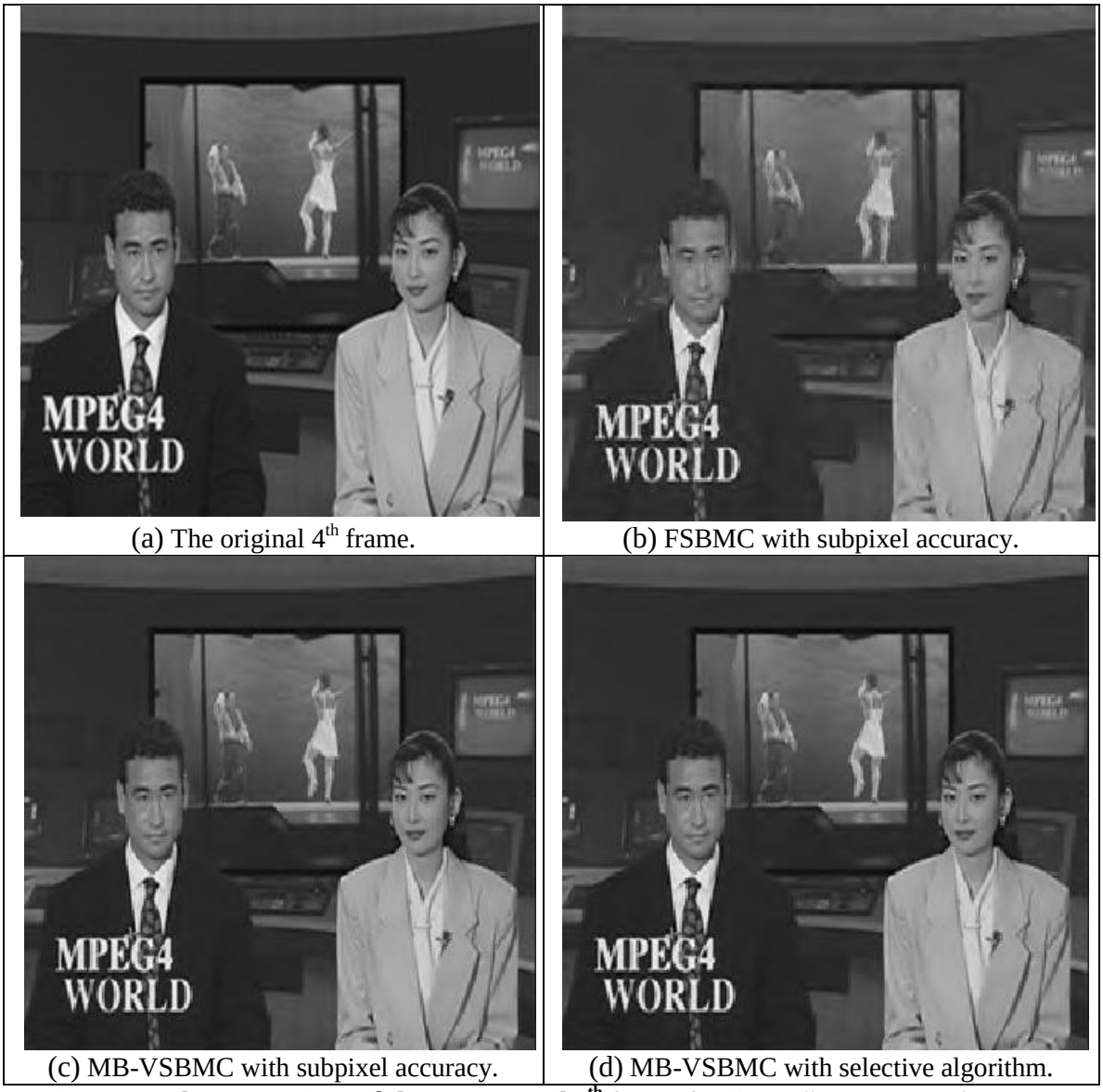
RDWT Domain	News		Forman	
	SSIM	PSNR	SSIM	PSNR
FSBMC	0.870	30.46	0.834	27.89
VSBMC	0.921	32.71	0.911	29.04
MB-VSBMC	0.978	33.65	0.923	30.47

Table 4.3 Comparison between conventional VSBMC, FSBMC and MB-VSBMC with sub-pixel accuracy.

RDWT Domain	News		Forman	
	SSIM	PSNR	SSIM	PSNR
FSBMC+Subpixel	0.884	32.02	0.866	29.70
VSBMC+Subpixel	0.941	34.93	0.927	31.27
MB-VSBMC+Subpixel	0.987	35.71	0.934	33.06

Table 4.4 Comparison between conventional VSBMC, FSBMC and MB-VSBMC with either a sub-pixel accuracy or selective algorithm.

RDWT Domain	News		Forman	
	SSIM	PSNR	SSIM	PSNR
FSBMC+Subpixel	0.884	32.02	0.866	29.70
VSBMC+Subpixel	0.941	34.93	0.927	31.27
MB-VSBMC+Subpixel	0.987	35.71	0.954	33.06
MB-VSBMC+Selective	0.986	35.56	0.942	32.81



(a) The original 4th frame. (b) FSBMC with subpixel accuracy.

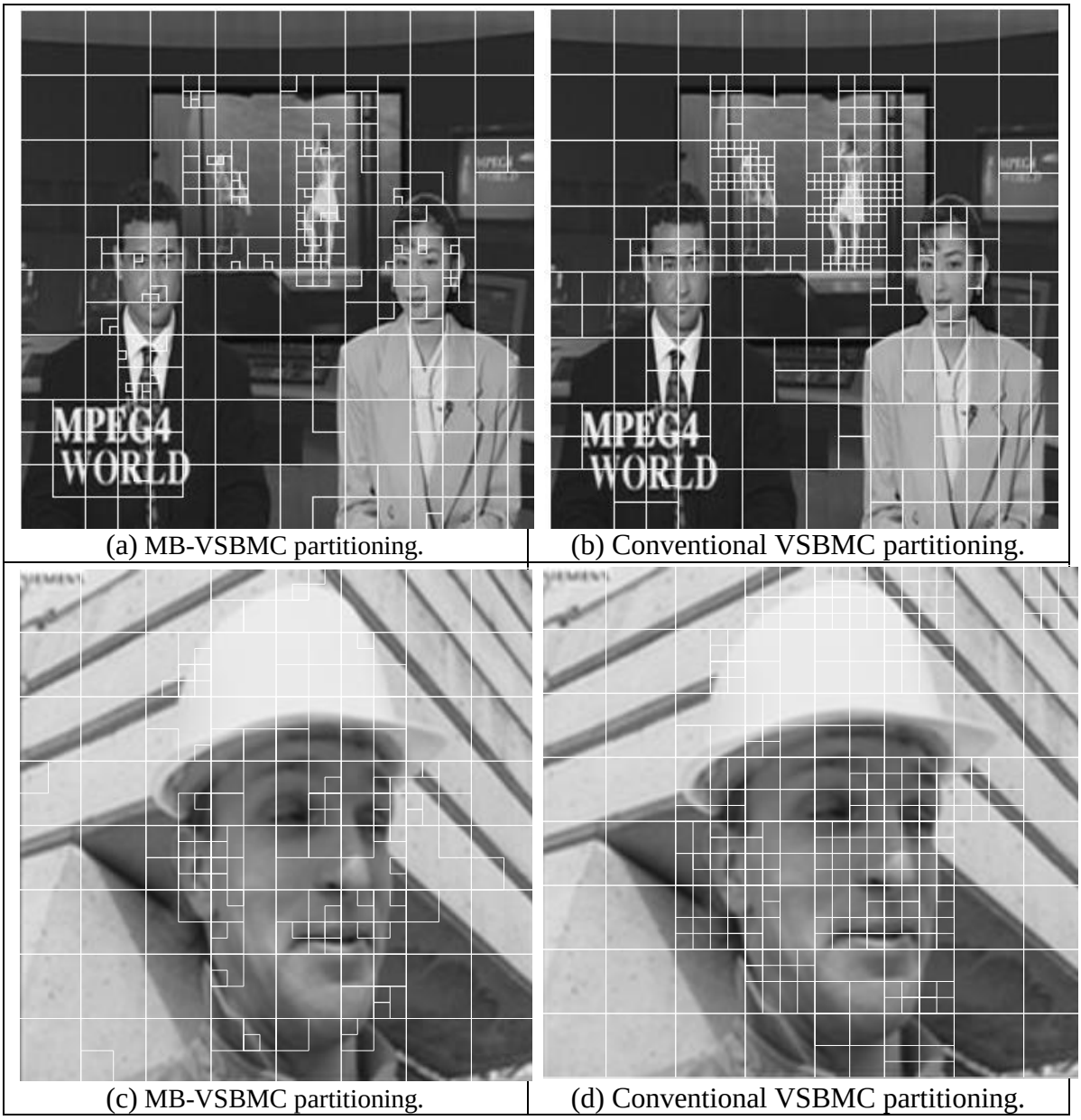
(c) MB-VSBMC with subpixel accuracy. (d) MB-VSBMC with selective algorithm.

Figure 4.6 The comparison of the compressed 4th frame for "News" sequence using three different block partitioning techniques.

(a) The original 6th frame. (b) FSBMC with subpixel accuracy.

(c) MB-VSBMC with subpixel accuracy. (d) MB-VSBMC with selective algorithm.

Figure 4.7 The comparison of the compressed 6th frame for "Foreman" sequence using three different block partitioning techniques.



(a) MB-VSBMC partitioning. (b) Conventional VSBMC partitioning.

(c) MB-VSBMC partitioning. (d) Conventional VSBMC partitioning.

Figure 4.8 An example of partitioning results using different approaches.

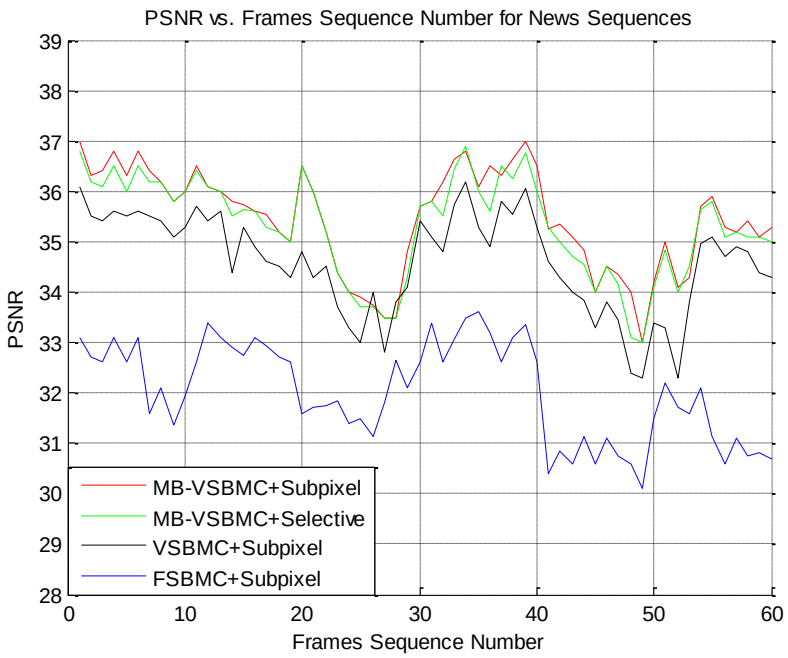


Figure 4.9 PSNR for "News" at 0.5 bpp for I and P, and 0.25 bpp for B frames.

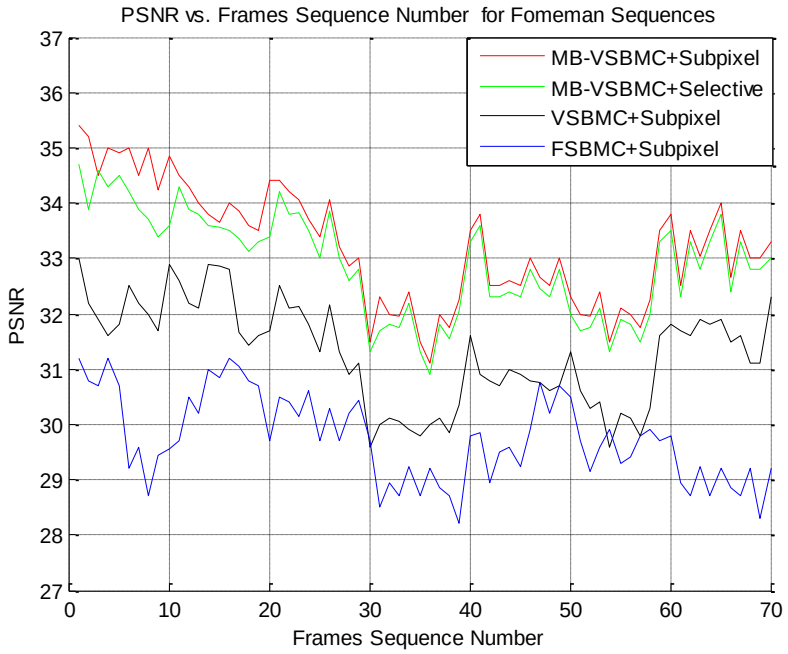


Figure 4.10 PSNR for "Forman" at 0.5 bpp for I and P, and 0.25 bpp for B frames.

CHAPTER 5

OPTIMIZED MULTIBAND VARIABLE BLOCK SIZE MOTION COMPENSATION APPROACH

5.1 Introduction

The MB-VSBMC approach discussed in the last chapter has achieved superior performance than the other methods. However, the approach itself is not optimized in the sense that those threshold parameters are chosen empirically. In this chapter, we will use the rate allocation theory to choose those parameters in an optimizing fashion.

5.2 Study of the Rate Allocation Theory

Efficient compression algorithms must minimize rate as well as distortion. A choice between different MVs or different block sizes is equivalent to a choice between points in the rate-distortion (R-D) curve. Using a Lagrange multiplier $\lambda \geq 0$, we can find points on the convex hull of all possible R-D pairs by solving the unconstrained problem [28].

$$\min_{B \in S} \{D(B) + \lambda \cdot R(B)\} \tag{5.1}$$

where *S* is the set of admissible bit allocations, and *D(B)* and *R(B)* are the total distortion and rate associated with the particular allocation $B \in S$ [29, 30]. Each convex hull point for $\lambda > 0$ is optimal in the sense that it has a lower distortion than any other possible R-D

pair having the same rate or less. By segmenting the image into *K* blocks, we can express the total bit usage *R* and distortion *D* as:

$$R(B) = \sum_{k=1}^{K} b_k \tag{5.2}$$

$$D(B) = \sum_{k=1}^{K} E_k(b_k) \tag{5.3}$$

where b_k is the number of bits used for coding the *k*-th block's motion representation and $E_k(b_k)$ is the block's resulting distortion. Combining (5.1) through (5.3), the unconstrained problem can be written as:

$$\min_{B \in S} \left\{ \sum_{k=1}^{K} \left[E_k(b_k) + \lambda \cdot b_k \right] \right\} \tag{5.4}$$

where each term may be minimized separately [28, 31]. The decisions can be made optimally by minimizing the objective function for each region *k*.

$$E_k(b_k) + \lambda \cdot b_k \tag{5.5}$$

This will sequentially, minimize $D(B) + \lambda \cdot R(B)$ over the entire image. This method is called the principle of separate minimization [28]. The minimizing of the objective function (5.5) for each separate region will result in a globally optimal solution for the unconstrained problem (5.1).

The Lagrange multiplier λ determines the relative importance of rate and distortion. For $\lambda = 0$, the distortion alone is minimized, resulting in a relatively high

rate. For positive values of λ, some increase in distortion is allowed, as long as it is accompanied by a saving in rate.

5.3 R-D Optimized FBMC

Consider a fixed size BMC. Denote the total distortion over the *k*-th block X_k using motion vector $(\Delta x, \Delta y)$ as $d_{X_k}(\Delta x, \Delta y)$; and assume that each motion vector can be represented by a variable-length codeword (e.g., from a Huffman code table) with a known number of bits $b_{X_k}(\Delta x, \Delta y)$.

$$(\Delta\tilde{x}, \Delta\tilde{y}) = \arg\min_{\Delta x, \Delta y} \left\{ d_{X_k}(\Delta x, \Delta y) + \lambda \cdot b_{X_k}(\Delta x, \Delta y) \right\} \tag{5.6}$$

This allows the optimized motion vector $(\Delta\tilde{x}, \Delta\tilde{y})$ to be chosen on the basis of rate and distortion, rather than distortion alone.

5.4 R-D Optimized VBMC

Consider a variable size BMC and assume the quadtree structure [33-35] is represented in this manner, i.e., two bits per merge/split decision, and that the leaf node block *X* is associated with a motion vector $(\Delta x, \Delta y)$, which uses $b_X(\Delta x, \Delta y)$ bits and has distortion $d_X(\Delta x, \Delta y)$.

Consider a $2^{l+1} \times 2^{l+1}$ sub-block X_{l+1} in the quadtree, composed of four adjacent $2^l \times 2^l$ sub-blocks $\{X_{l,m}, m = 1,2,3,4\}$ at level $l \geq l_0$ in the quadtree. Assume the optimal

sub-tree structure is known for each of the four sub-blocks, and that optimal motion vectors are known for each leaf node. Denote the total bit usage and distortion of these optimal constituent sub-trees as $b^*(X_{l,m})$ and $d^*(X_{l,m})$. Next, find an optimal motion vector for the entire block X_{l+1} and its incurred bit usage $b_{X_{l+1}}(\Delta\tilde{x},\Delta\tilde{y})$ and distortion $d_{X_{l+1}}(\Delta\tilde{x},\Delta\tilde{y})$. Using the principle of separate minimization (5.4) [28], the subtrees should be combined into a single leaf node whenever

$$d_{X_{l+1}}(\Delta\tilde{x},\Delta\tilde{y}) + \lambda \cdot b_{l+1}^{T}(X_{l+1}) \le \sum_{m=1}^{4} d^*(X_{l,m}) + \lambda \cdot b_l^{T}(X_{l+1}) \tag{5.7}$$

The above condition can be expressed in a simpler form whenever

$$\Delta d \le \lambda \cdot \Delta b \tag{5.8}$$

where Δd is the error reduction and Δb is the increment in the coding bits under the condition that a block is to be divided. If the MVs are assumed to be fixed-length coded, and each MV required B_{mv} bits, then the total coding bits representing the motion information can be expressed as in [36]:

$$B = nB_{mv} + C_{qt} \tag{5.9}$$

where *n* is the number of leaves in the tree and C_{qt} is the number of bits used for coding a tree. Since each splitting operation produces three additional blocks, therefore, three more motion vectors are added. The splitting condition stated above can be rewritten as

$$D_j^l - (D_{4j}^{l+1} + D_{4j+1}^{l+1} + D_{4j+2}^{l+1} + D_{4j+3}^{l+1}) \le \lambda(3B_{mv} + \Delta C_{qt}) \quad (5.10)$$

where D_j^l represents the prediction error of the *j-th* block in level *l*. ΔC_{qt}, is the increment in the number of bits due to the partitioning of the block. Usually ΔC_{qt} is equal to 4 bits.

5.5 Rate-Distortion Curve

The unconstrained rate and distortion values *R(*λ) and *D(*λ*)* are monotonic in the Lagrange multiplier λ. As λ is swept through, all the convex hull points of the composite *R-D* curve are traced out [66]. Thus λ could be interpreted as a quality index as it is swept from 0 (highest rate, lowest distortion) to ∞ (lowest rate, highest distortion). Therefore, the unconstrained problem becomes the minimization of the Lagrangian cost function $J(\lambda)$ defined as:

$$J(\lambda) = D(\lambda) + \lambda \cdot R(\lambda) \quad (5.11)$$

All signal block combinations must be considered at a slope point λ on their R-D curves for a given $\lambda = |\Delta D/\Delta R|$. Figure 5.1 shows an example of a composite R-D curve with combination choices for the convex-hull points, with optimal tree structure for a given budget constraint. In the example, we calculate the distortion and its associate bit rate for each block-splitting combination, and then we pick the best splitting combination that has not exceeded the bit rate budget line, and has simultaneously minimized both the distortion and the bit rate. More details on R-D curve can be found in [66]. In this

chapter, we applied the rate allocation theory to our proposed MB-VSMC algorithm by using the rate-distortion curve to optimize the threshold values for the splitting process.

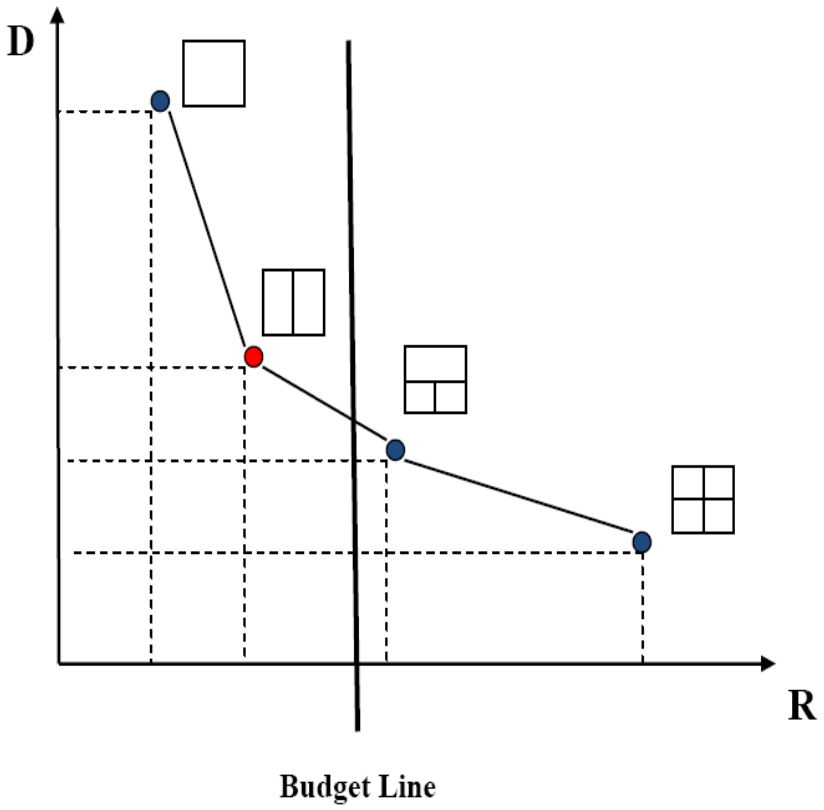


Figure 5.1 Example of a composite R-D curve. Each square on the convex hull points represents a potential configuration for block partitions.

5.6 Distortion Measurement in RDWT

In a J-scale RDWT decomposition, each $N \times N$ block in the original spatial domain corresponds to $3j+1$ blocks of the same size, one in each subband [37]. The collection of these co-located blocks is called a set. Each set contains all the different

phases of RDWT coefficients. In the ME procedure, block matching is used to determine the motion of each set as a whole. Specifically, a block-matching procedure uses a cross-subband distortion measure that sums absolute errors for each block of the set. The coefficients from all phases in both current and reference frames contribute to the distortion measurement [65]. Therefore, the mean absolute distortion (MADIST) can be obtained using,

$$MADIST(x, y, \Delta_x, \Delta_y) = \frac{1}{N^2} \sum\sum AE(x + k, y + l, \Delta_x, \Delta_y) \tag{5.12}$$

The absolute error (AE) is

$$AE(x, y, \Delta_x, \Delta_y) = \frac{1}{2}\left\{\begin{array}{l} \left|V_1^{cur}(x, y) - V_1^{ref}(x + \Delta_x, y + \Delta_y)\right| + \left|H_1^{cur}(x, y) - H_1^{ref}(x + \Delta_x, y + \Delta_y)\right| + \\ \left|D_1^{cur}(x, y) - D_1^{ref}(x + \Delta_x, y + \Delta_y)\right| + \left|B_1^{cur}(x, y) - B_1^{ref}(x + \Delta_x, y + \Delta_y)\right| \end{array}\right\} \tag{5.13}$$

where *cur* and *ref* denote subbands from the current and reference frames, respectively, and B_j , H_j , V_j , and D_j are the baseband, horizontal, vertical, and diagonal subbands, respectively, at scale j [65].

5.7 R-D Optimized MB-VBMC and Decision Criterion

We apply the rate-distortion theory to our new MB-VBMC approach. The whole process is again applied in the following five steps similar to those described in chapter four. The first three steps are basically identical to those not optimized before. They are

listed below for reference. However, starting in the fourth step the concept of rate allocation theory is integrated into the splitting procedure.

5.7.1 Splitting Process Using Rate Allocation Theory

First: Create all phase correlation edge mask. The correlation edge mask acts as a map for the decision making of the variable block size, since it highlights the edges.

$$\text{mask}(x,y) = \left|\prod_{j=J_0}^{J_1} V_j(x,y)\right| + \left|\prod_{j=J_0}^{J_1} H_j(x,y)\right| + \left|\prod_{j=J_0}^{J_1} D_j(x,y)\right| \tag{5.14}$$

where J_0 and J_1 are the starting and ending scales, respectively, of the correlation operation. Note that mask(x, y) is the resulting correlation image with the same dimension as the original image. We will use all-phase correlation edge mask of the current frame to determine which 16×16 MB is the candidate to be split by setting a number of thresholds.

Second: determine the global maximum of the mask,

$$\text{Mask}_{\max} = \max(\text{mask}(x,y)) \tag{5.15}$$

and set the threshold, τ as

$$\tau = \alpha \cdot \text{Mask}_{\max} \tag{5.16}$$

where the threshold parameter is $\alpha, 0 \le \alpha \le 1$.

Third: Divide the current correlation edge mask into 16×16 MBs and select each MB with a value larger than the threshold.

$$Block_{Avg}^{16\times16} \geq \tau \tag{5.17}$$

Fourth: For the chosen MB, divide the current and the reference correlation masks into 8×8 MBs and subtract the co-located 8×8 MBs from each other and then test the result against the correlation threshold τ_1 .

$$Diff_{i,j} = \sum_{i,j}^{8} |\,\mathrm{mask}_{i,j}^{cur}\,| - |\,\mathrm{mask}_{i.j}^{ref}\,| \tag{5.18}$$

where $Diff_{i,j}$ is the absolute difference between two masks. *i* and *j* are the horizontal and vertical displacements. $\mathrm{mask}_{i,j}^{cur}$ and $\mathrm{mask}_{i.j}^{ref}$ are co-located 8×8 MBs for the current and reference frame, respectively.

Thus, set the threshold τ_1 as:

$$\tau_1 = \begin{cases} \left\lfloor \dfrac{Diff_{i,j}^{\max} + Diff_{i,j}^{\min}}{2} \right\rfloor & \text{if } \Delta d < \lambda \cdot \Delta b \\ \varphi \cdot \left\lfloor \dfrac{Diff_{i,j}^{\max} + Diff_{i,j}^{\min}}{2} \right\rfloor & \text{Otherwise} \end{cases} \tag{5.19}$$

where Δd is the prediction error reduction and Δb is the increment in the coding bits under the condition that a block is to be divided. The prediction error can be expressed as

$$\Delta d = D_j^l - \sum_{n=1}^{4} D_{4j+n}^{l+1} \tag{5.20}$$

where D_j^l represents the prediction error of the *j-th* block in level *l*, D_{4j+n}^{l+1} represents the prediction error of the *j-th* block in sub-level *l+1*; and *n* is the number of leaves in the tree. If the motion vectors are assumed to be fixed-length coded and each vector requires B_{mv} bits [6], then the total number of coding bits representing the motion information can be expressed as:

$$\Delta b = nB_{mv} + C_{qt} \tag{5.21}$$

where *n* is the number of leaves in the tree and C_{qt} is the number of bits used for coding a tree. The parameter $\varphi > 1$ will be iterated until the threshold value of τ_1 satisfies the optimization condition $\Delta d > \lambda \cdot \Delta b$.

Fifth: The selected 8×8 MBs can be split again using the same procedures and equations above by changing the index 8×8 to 4×4 in block size.

5.8 Experimental Results

For the experiment, we use 60 frames of 352×288 "News" sequence (CIF); and 70 frames of 144×176 "Foreman" sequence (QCIF). The sequences are grayscaled and have a temporal sampling of 25 frame/sec. The first frame is intra-encoded (I-frame) while all subsequent frames use ME/MC (P and B-frames). All wavelet transforms (RDWT) use the Daubechies 9-7 filter with symmetric extension and a decomposition of J = 2 level. The parameters α, β and α_n are 0.4, 0.68 and 0.73 respectively. For comparison purposes, we use the peak signal-to-noise ratio (PSNR) and the structural similarity index (SSIM) [39]. The PSNR and SSIM values were calculated for the coding system and shown in

Tables 5.1. The results include our proposed MB-VSBMC and conventional FSBMC method (8×8 block size), in addition to the conventional VSBMC [40] by replacing the CODEC algorithm from DCT to SPIHT and applying decision criteria to the wavelet approximation band. By comparing Table 5.1 to Table 4.4, one can see that approaches using R-D optimization have better PSNR and SSIM values.

Table 5.1 Comparison between conventional VSBMC, FSBMC and MB-VSBMC with either a sub-pixel accuracy or selective algorithm. The R-D optimization method is applied to all algorithms.

RDWT Domain	News		Forman	
	SSIM	PSNR	SSIM	PSNR
FSBMC+Subpixel	0.887	32.72	0.867	30.42
VSBMC+Subpixel	0.944	35.56	0.928	31.55
MB-VSBMC+Subpixel	0.988	36.14	0.956	33.78
MB-VSBMC+Selective	0.987	35.97	0.946	32.92

We also did a study of the number of blocks used for each algorithm for the optimized approach. For example, the comparison between the conventional VSBMC algorithm and the proposed algorithm MB-VSBMC algorithm is shown in Figure 5.2. The MB-VSBMC algorithm reduces the number of the blocks used for conventional VSBMC by about 8-11%.

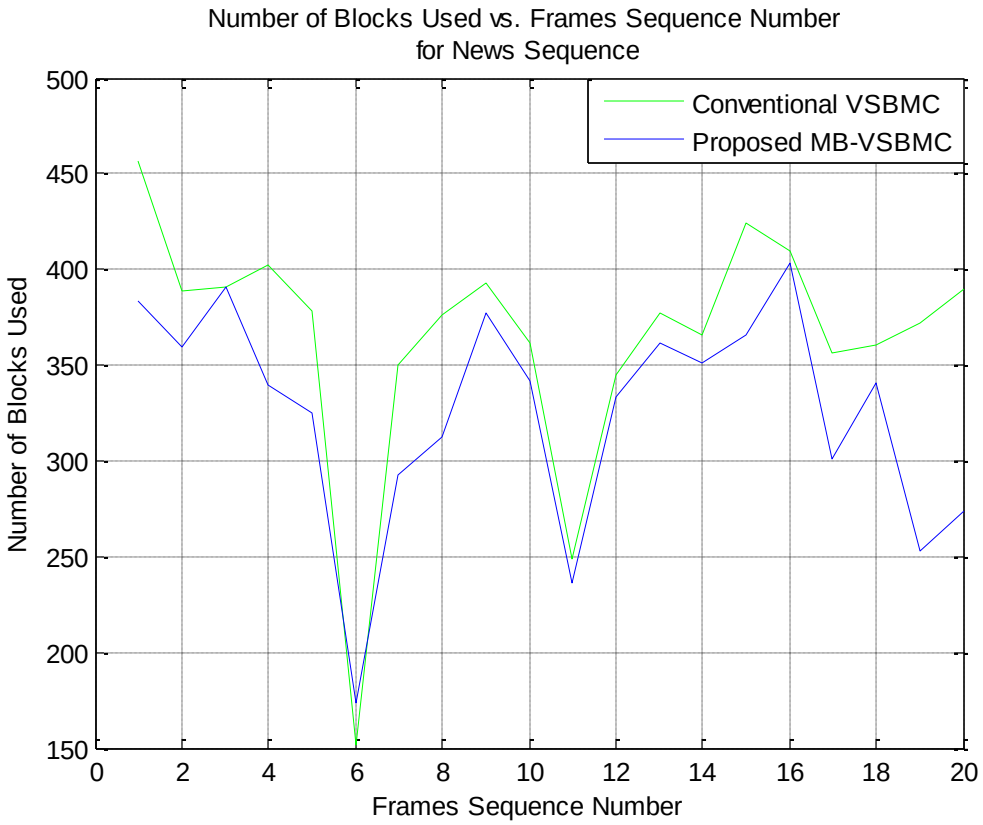


Figure 5.2 The number of the blocks used vs. frames sequence number for "News" sequence.

CHAPTER 6

OVERLAPPED BLOCK MATCHING IN REDUNDANT WAVELET DOMAIN

6.1 Introduction

Overlapped block-matching motion compensation (OBMC) is an enhancement to the conventional block matching algorithm, which allows the blocks in the grid to overlap. The pixel intensity for a pixel in a reconstructed frame is not only derived by translating a single pixel from the reference frame, but is also affected by translating pixels according to the MVs of neighboring blocks [67]. When the blocks overlap, the pixel intensities are combined linearly, with the weights taken from a window function over the block.

6.2 Overlapped Block Motion Compensation

In a conventional block-based motion prediction (Figure 6.1), each block is motion-compensated independently of other blocks. Consequently, the motion vector for a given block is not necessarily the same as the vectors of its adjacent blocks, even though it is likely that the motion of the neighboring blocks is similar. This disparity causes discontinuity among consecutive blocks in the motion-compensated frame, a major cause of blocking artifacts. To mitigate this effect, the OBMC approach was proposed in [41].

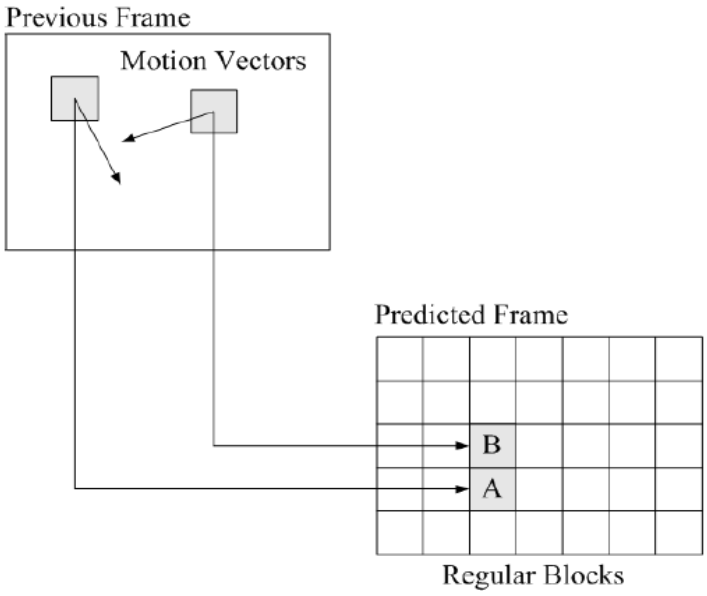


Figure 6.1 Conventional block motion compensation.

Figure 6.2 shows an OBMC approach, in which a weighted sum of multiple predictions is used to motion-compensate each block. Let $P_i(x, y)$ be a prediction of the current block obtained from a reference block, which is weighted by matrix $W_i(i, j)$. In OBMC, the P_i predictions of the current block are generated by using the motion vectors of neighboring blocks. Thus, the weighted prediction is the following [42-45],

$$\tilde{P}_i(x, y) = P_i(x, y) \times W_i(i, j) \tag{6.1}$$

where × represents element-by-element multiplication. The final prediction of the current block is the following:

$$P(x, y) = \sum_i \tilde{P}_i(x, y) \tag{6.2}$$

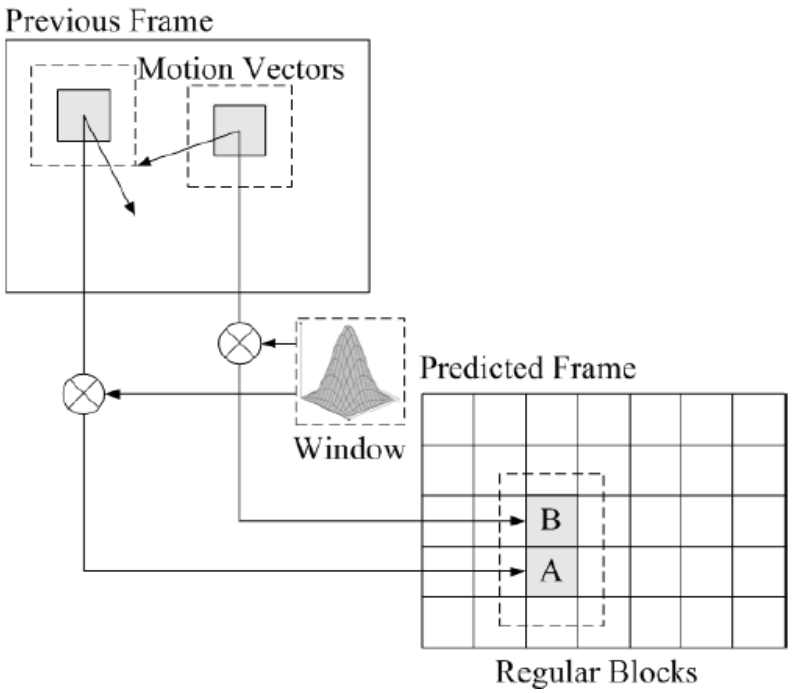


Figure 6.2 Overlapped block motion compensation.

6.3 Weight Windows Selection

There are several types of weight windows that can be used for OBMC, and different windows have different performances. The three popular overlapped windows are the trapezium window, bilinear window, and raised cosine window [47], as shown in Figure 6.3. The trapezium window is recommended in the H.263 standard [2]. Its weights are based on experience. The values of the weights at the four corners are all zeros. The weight values of the bilinear window are cone-like and those of the raised cosine window are very smooth. Expressions for the weight values for the raised cosine and the bilinear types of windows are as follows. Raised cosine: $W(x, y) = w_x \cdot w_y$

$$w_x = w_y = \frac{1}{2}\left[1 - \cos\frac{\pi(x+1/2)}{N}\right] \qquad \text{for } x = 0,...,2N-1 \tag{6.3}$$

Bilinear: $W(x, y) = w_x \cdot w_y$

$$w_x = w_y = \begin{cases} \frac{1}{N}(x+1/2) & \text{for } x = 0,...N-1 \\ w_{2N-1-x} & \text{for } x = N,...2N-1 \end{cases} \tag{6.4}$$

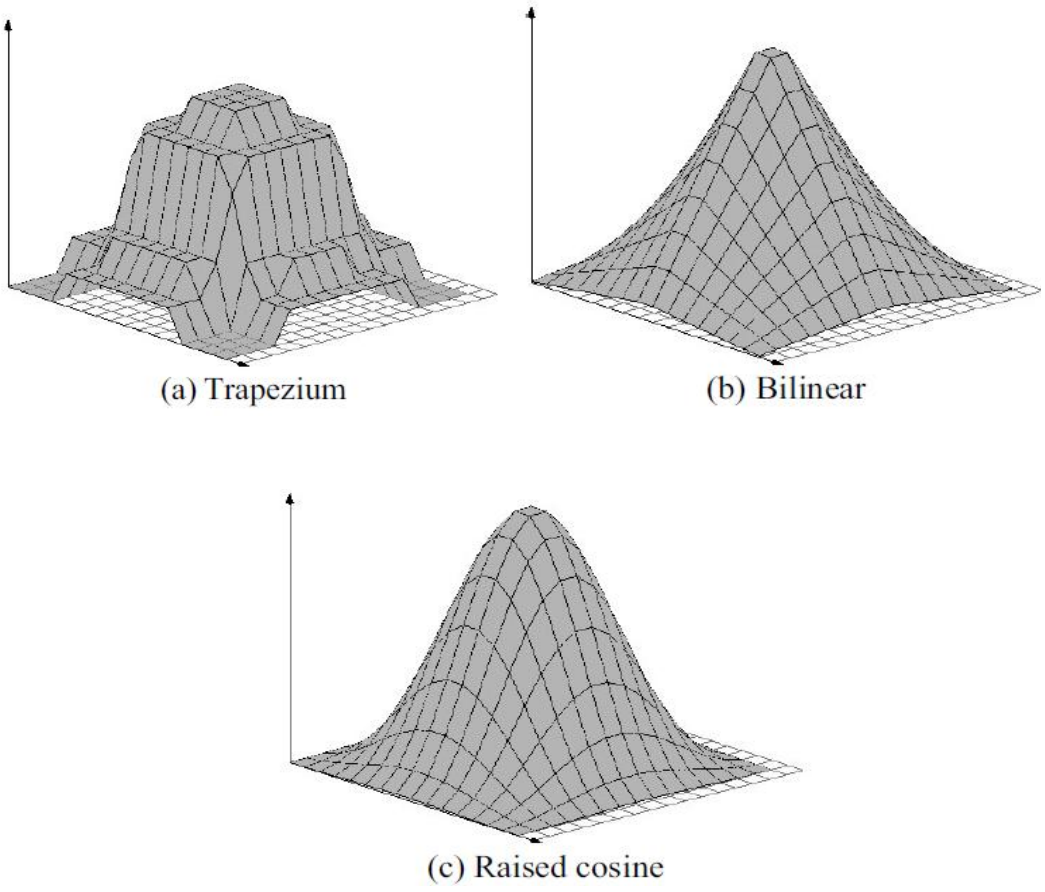


Figure 6.3 Three OBMC weight windows.

6.4 OBMC Implementation in RDWT

The implementation of OBMC in the RDWT is a straightforward adaptation of OBMC in the spatial domain. It is well known that OBMC in the spatial domain can increase performance greatly; thus, it has been adopted in the H.263 standard. Since RDWT coefficients retain the "spatial coherence" of the original image; therefore, OBMC in the RDWT domain is straightforward application. Since there are *3J +1* subbands for a *J*-scale decomposition, we must deploy OBMC in all the subbands in the RDWT domain following the same procedure. The research in this section uses the trapezium weight window to obtain the weighted prediction MBs.

OBMC in RDWT domain uses a weighted sum of multiple predictions to motion-compensate each block. Let $P_i(x, y)$ be a prediction of the current block obtained from a reference block. Then, the P_i predictions of the current block are generated by using the motion vectors of itself and neighboring blocks. The weighted prediction is the following,

$$\tilde{P}_i(x, y) = P_i(x, y) \times W_i(i, j) \tag{6.5}$$

The final prediction of the current block is the following:

$$P(x, y) = \sum_i \tilde{P}_i(x, y) \tag{6.6}$$

In each subband, we define 8×8 MBs which are further divided into four 4×4 sub-MBs. As illustrated in Figure 6.4, four sub-MBs within the current MB and the neighboring eight MBs are used to form a prediction of the current MB.

The prediction of the selected sub-MB within the current MB, such as V_A block, will be formed using the weighted sum of three MBs; obtained through the motion vector for the current MB (Δ_x, Δ_y), plus the motion vectors of the two nearest neighboring MBs, one from the vertical direction (Δ_x^V, Δ_y^V) and one from the horizontal direction (Δ_x^H, Δ_y^H), as shown in Figure 6.4. Depending on the different locations of those prediction blocks, there are three 8×8 matrices of weighting values as illustrated in Figures 6.5 - 6.7. Those weighting matrices are explained in detail in [41, 65, and 67]. The prediction of $P(x, y)$ is of 8×8 block size, and can be obtained using,

$$P(x,y) = \left[\tilde{P}(x,y) + \tilde{P}_V(x,y) + \tilde{P}_H(x,y)\right] / C \tag{6.7}$$

where C is a constant such as three or eight. The expression $p(x+\Delta_x^k, y+\Delta_y^k)$ refers to the prediction value at the position $(x+\Delta_x^k, y+\Delta_y^k)$ in the reference frame. The notation k can be null, H or V. Consequently, the weighted prediction for V_x is the following:

$$\tilde{P}(x,y) = p(x+\Delta_x, y+\Delta_y) \times W_x(i,j) \tag{6.8}$$

The weighed prediction for the vertical displacement is the following:

$$\tilde{P}_V(x,y) = p(x+\Delta_x^V, y+\Delta_y^V) \times W_V(i,j) \tag{6.9}$$

The weighed prediction for the horizontal displacement is the following:

$$\tilde{P}_H(x,y) = p(x+\Delta_x^H, y+\Delta_y^H) \times W_H(i,j) \tag{6.10}$$

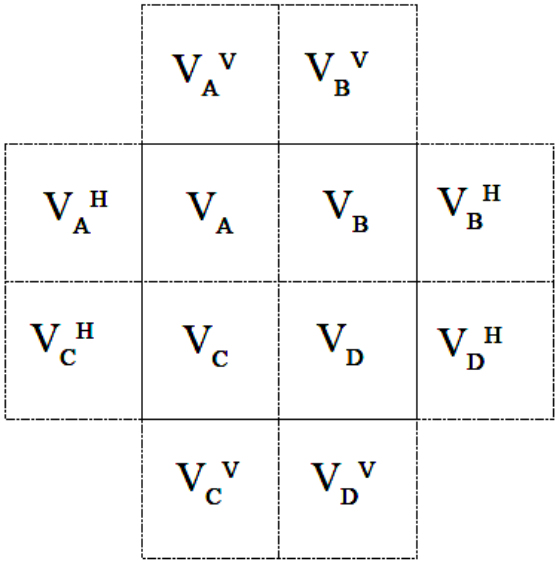


Figure 6.4 The Block V_x is predicted using the MV for block V_x plus the MVs for blocks V_x^V and V_x^H. The notation *x* can be A, B, C or D.

4	5	5	5	5	5	5	4
5	5	5	5	5	5	5	5
5	5	6	6	6	6	5	5
5	5	6	6	6	6	5	5
5	5	6	6	6	6	5	5
5	5	6	6	6	6	5	5
5	5	5	5	5	5	5	5
4	5	5	5	5	5	5	4

Figure 6.5 Weighting values W_x, for prediction with motion vector of current block.

2	2	2	2	2	2	2	2
1	1	2	2	2	2	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	2	2	2	2	1	1
2	2	2	2	2	2	2	2

Figure 6.6 Weighting values W_V , for prediction with motion vectors of the blocks on top or bottom of current block.

2	1	1	1	1	1	1	2
2	2	1	1	1	1	2	2
2	2	1	1	1	1	2	2
2	2	1	1	1	1	2	2
2	2	1	1	1	1	2	2
2	2	1	1	1	1	2	2
2	2	1	1	1	1	2	2
2	1	1	1	1	1	1	2

Figure 6.7 Weighting values W_H , for prediction with motion vectors of the blocks to the left or right of current block.

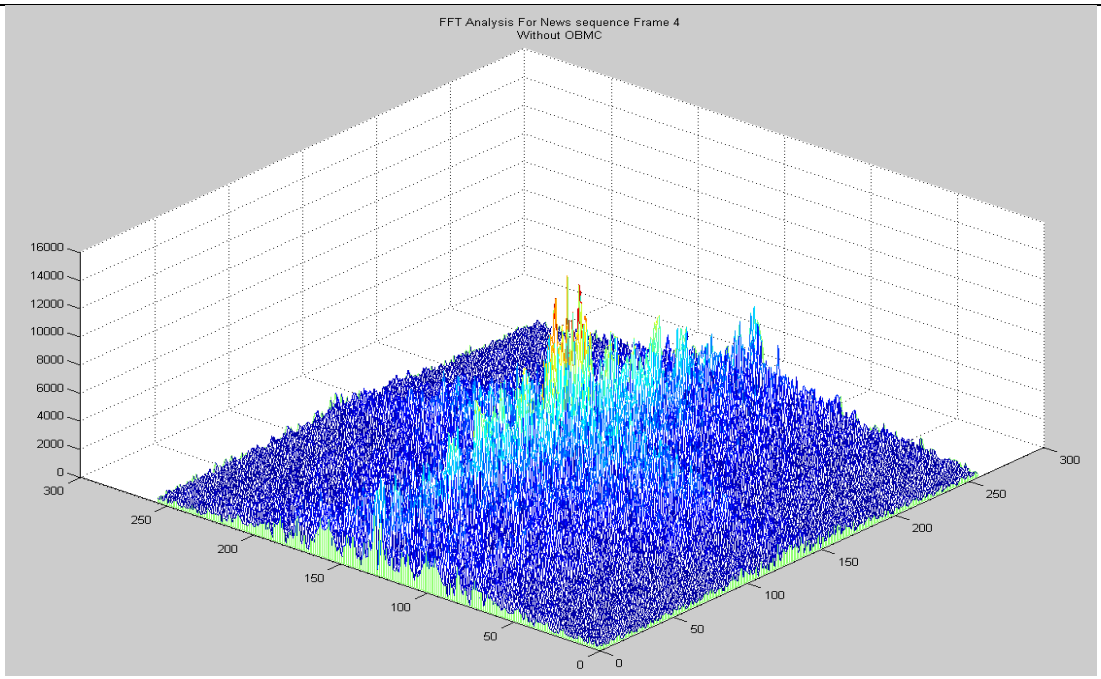
6.5 Experimental Results

For the experiment, we use 60-frames of 352×288 "News" sequence, with common intermediate format CIF (standard video format used in videoconferencing); and 70-frames of 144×176 "Foreman" sequence, with quarter common intermediate format

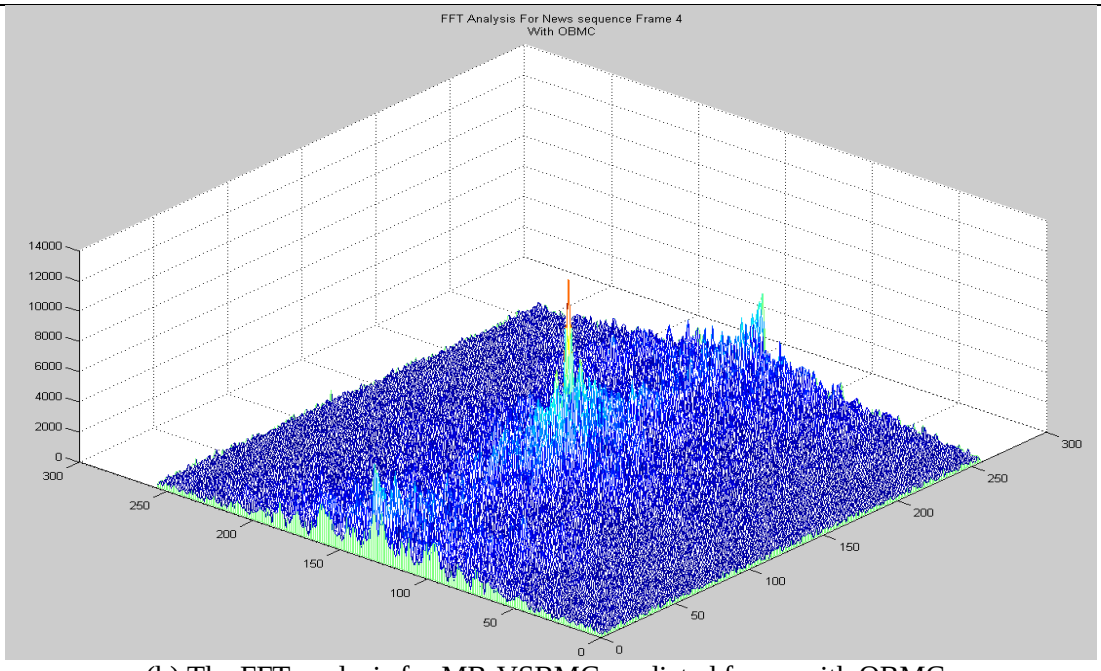
(QCIF). The sequences are grayscaled and have a temporal sampling of 25 frame/ sec. The first frame is intra-encoded (I-frame) while all subsequent frames use ME/MC (P and B-frames). All wavelet transforms (RDWT) use the Daubechies 9-7 filter with symmetric extension and a decomposition of $J = 2$ level. The parameters α, β and α_n are 0.4, 0.68 and 0.73 respectively. The core compression engine in all experiments is SPIHT. The SPIHT produces an embedded coding. Each frame of the sequence is coded at exactly the specified target rate with compression rate of 0.5 bpp for I frame and 0.25 bpp for P and B frames.

Figure 6.8 shows the fast Fourier transform (FFT) analysis of the predicted frame. Figure 6.8.a shows the FFT analysis for MB-VSBMC predicted frame without OBMC algorithm. Figure 6.8.b shows the FFT analysis for MB-VSBMC predicted frame with OBMC algorithm. In the comparison between (a) and (b), it is observed that the number of high frequency components (which represent more energy) is reduced. This means that the prediction error is also reduced. Figure 6.9 shows an example of the OBMC effect on the blocking edge artifacts. By comparing Figure 6.9.a and Figure 6.9.b, it shows that OBMC has reduced the blocking edge artifacts and has improved the quality.

Table 6.1 shows the comparison in terms of PSNR and SSIM values for OBMC applied to the approaches of conventional VSBMC, FSBMC, and MB-VSBMC with either sub-pixel accuracy or selective algorithm. When we compare the results between Table 6.1 and Table 4.4, we can conclude that the OBMC algorithm produces better compression quality. Usually, the OBMC approach is better than the non-OBMC approach by about one to two dB.



(a) The FFT analysis for MB-VSBMC predicted frame without OBMC.



(b) The FFT analysis for MB-VSBMC predicted frame with OBMC.

Figure 6.8 FFT analysis for OBMC-related predicted frames.

(a) MB-VSBMC predicted frame without OBMC

(b) MB-VSBMC predicted frame with OBMC

Figure 6.9 OBMC effect on the blocking edge artifact.

Table 6.1 OBMC comparisons between conventional VSBMC, FSBMC and MB-VSBMC with either a sub-pixel accuracy or selective algorithm.

RDWT Domain	News		Forman	
	SSIM	PSNR	SSIM	PSNR
FSBMC+Subpixel	0.891	33.82	0.875	31.68
VSBMC+Subpixel	0.952	36.32	0.938	32.45
MB-VSBMC+Subpixel	0.989	37.23	0.967	34.96
MB-VSBMC+Selective	0.988	36.91	0.953	34.07

CHAPTER 7

THREE-DIMENSIONAL VIDEO COMPRESSION IN REDUNDANT WAVELET DOMAIN

7.1 Introduction

The 3D video technology enables various views to be integrated into a single 3D video system. Specifically, in 3D-TV video applications, several 3D video systems have been introduced in [51-57]. They can be classified into two classes with respect to the amount of employed 3D geometry. A first class of 3D video systems is based on multiple texture views of the video scene, called N-texture representation format. The N-texture approach forms the basis for the emerging multi-view video coding (MVC) standard currently developed by the Joint Video Team (JVT) [52]. Figures 7.1 and 7.2 have some illustrations on this.

However, due to the significant amount of data to be stored, the main challenge of the MVC standard is to define efficient coding and decoding tools. To this end, a number of H.264/MPEG-4 AVC coding tools have been proposed and evaluated within the MVC framework. The first coding tool exploits the similarity between the views by multiplexing the captured views and encoding the resulting video stream by a modified H.264/MPEG-4 AVC encoder [53, 54]. The second coding tool equalizes the inter-view illumination to compensate for mismatches across the views captured by different cameras [55]. The latest description of the standard can be found in the Joint Draft 8.0 on Multi-view Video Coding [51].

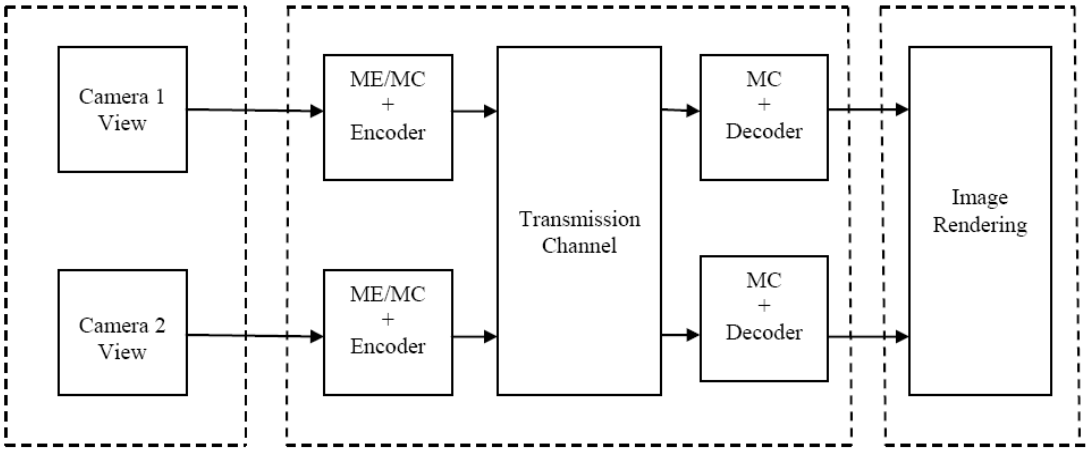


Figure 7.1 The N-texture Multi-view Video Coding (MVC).

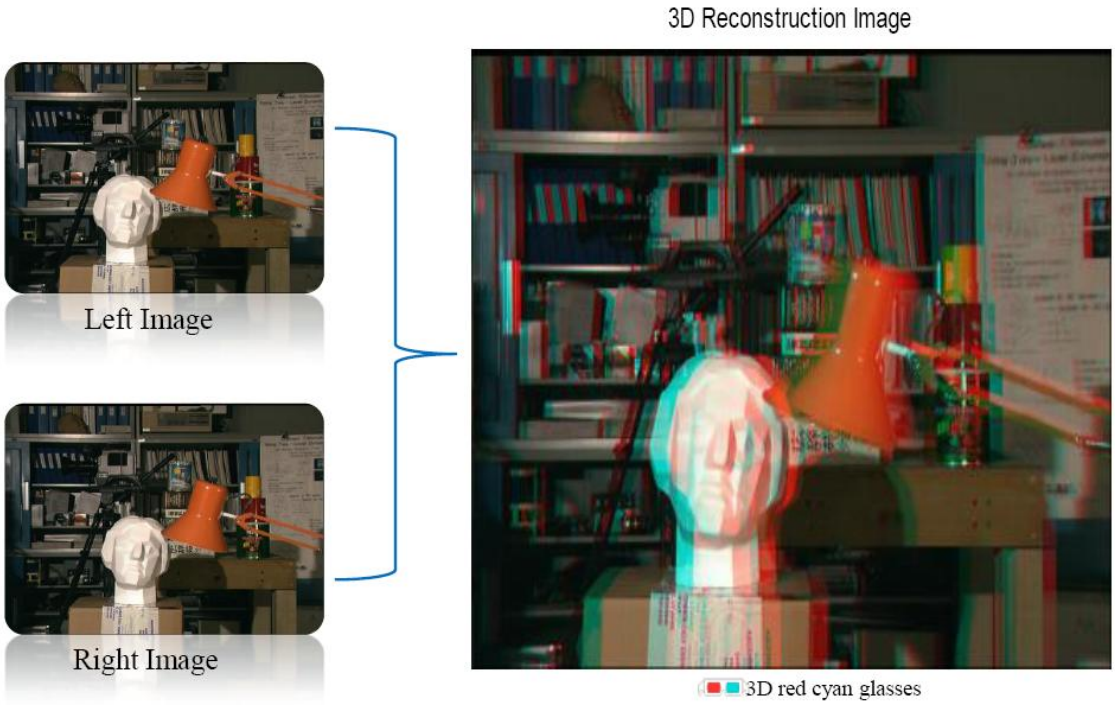


Figure 7.2 Example of the N-texture Multi-view Video Coding (MVC).

One advantage of the above-mentioned N-texture representation format is that no 3D geometric description of the scene is required. Because 3D geometry is not used, this 3D video format allows a simple video processing chain at the encoder. However, such a 3D video representation format involves a high complexity decoder for the following reason [56]; a multi-view display supports a varying number of views at the input, which makes it impractical to prepare these views prior to transmission. Instead, intermediate views should be interpolated from the transmitted reference views at the decoder, where the display characteristics are known. To obtain high-quality interpolated views, a 3D geometric description of the scene is necessary, thereby involving computationally expensive calculations at the receiver side.

A second class of 3D video systems relies on a partial-3D geometric description of the scene [57]. The scene geometry is typically described by a depth map, or depth image, that specifies the distance between a point in the 3D world and the camera. Typically, a depth image is estimated from two images by identifying corresponding pixels in the multiple views; in other words, the point-correspondences that represent the same 3D scene point. Using depth images, new views can be subsequently rendered or synthesized using a depth Image based rendering (DIBR) algorithm. Here, the term DIBR corresponds to a class of rendering algorithms that use depth and texture images simultaneously to synthesize virtual images. Considering a 3D-TV application, it is assumed that the scene is observed from a narrow field of view (short baseline distance between cameras). As a result, a combination of only one texture and one depth video sequence is sufficient to provide appropriate rendering quality. The 1-depth/1-texture

approach was recently standardized by Part 3 of the MPEG-C video specification [58-61]. This system is illustrated in Figure 7.3. The different approaches to video compression explained in previous chapters will be applied to texture image and depth image.

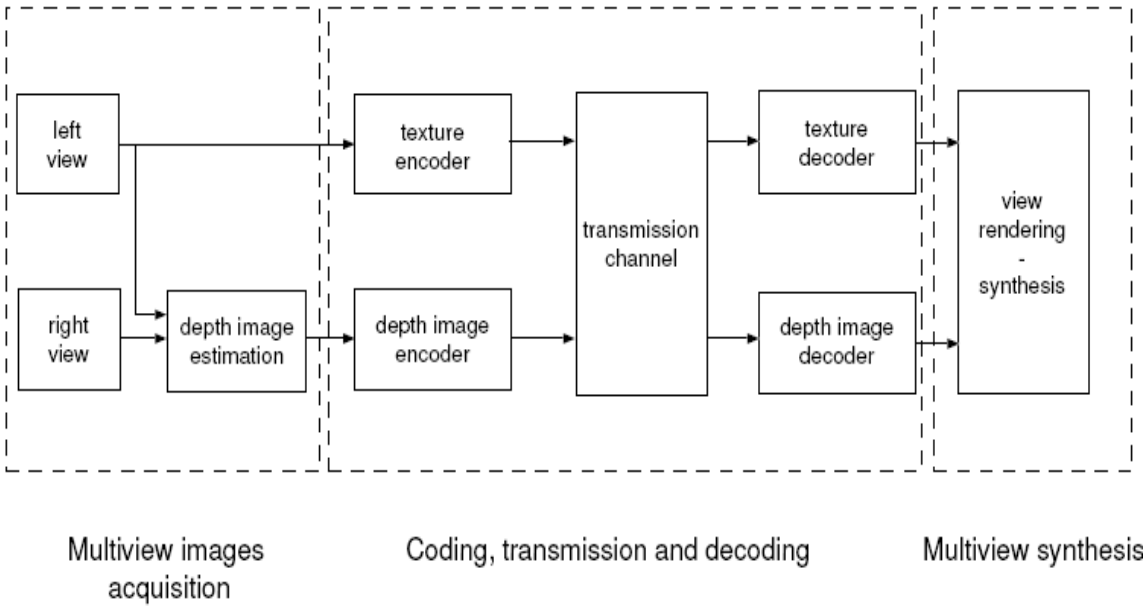


Figure 7.3 1-depth/1-texture multiview video compression system.

7.2 Stereo Constraints/ Epipolar Constraint

When images of a scene are captured using two cameras simultaneously, these cameras are termed a stereo-pair and produce stereo-pairs of images. The properties of cameras so configured are determined by their epipolar geometry, which describes the relationship between world points observed in their fields of view and the images

imposing on their respective sensing planes. The image-plane locations of each world point as sensed by the camera pair are called corresponding or matched points [60, 61]. Corresponding points within stereo-pair images are connected by the fundamental matrix. If known, it provides fundamental information on the epipolar geometry of the stereo-pair setup. However, finding corresponding points between images is not a trivial task. There are many factors which can confound this process, such as occlusions, limited image resolution and quantization, distortions, noise and many others. Technically, matching is said to be under-constrained; in other words, there is not sufficient information available within the compared images to guarantee finding a unique match. However, matching can be made easier by applying a set of rules known as stereo constraints, of which the most important is the epipolar constraint, and this implies that corresponding points always lie on corresponding epipolar lines [62].

The epipolar constraint limits the search for corresponding points from the entire 2D space to a 1D space of epipolar lines. Although the positions of the epipolar lines are not known in advance; in the special case when stereo-pair cameras are configured with parallel optical axes called the canonical or standard stereo system, the epipolar lines follow the image (horizontal) scan-lines. The problem of finding corresponding points is one of the essential tasks of computer vision. Figure 7.4 shows the epipolar geometry for parallel pin-hole cameras.

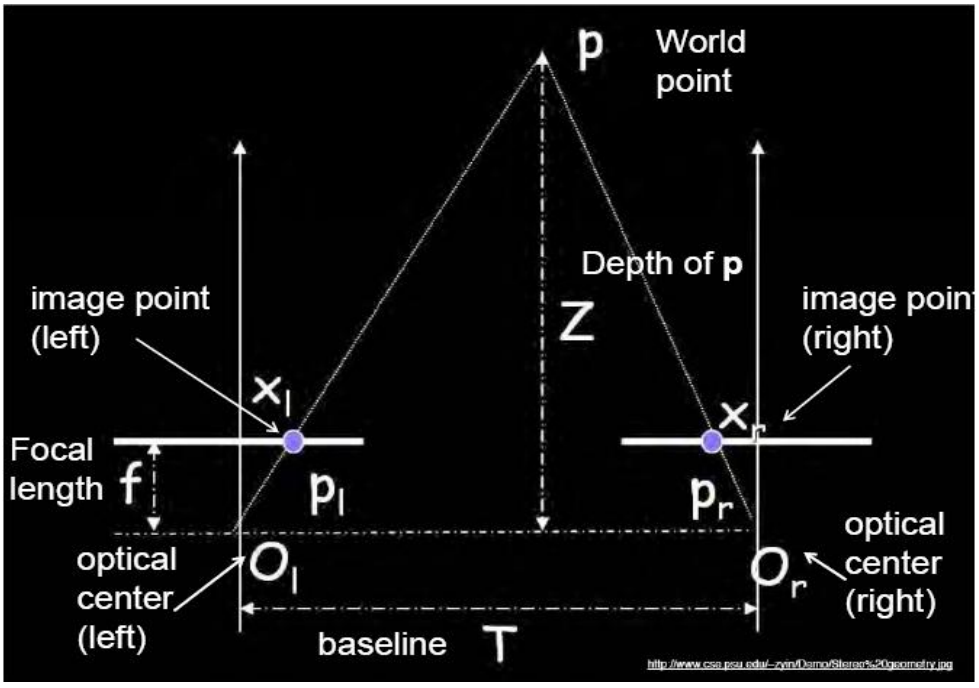


Figure 7.4 Epipolar geometry.

In Figure 7.4, left epipole is the projection of O_r on the left image plane. Right epipole is the projection of O_l on the right image plane. Epipolar plane is the plane defined by P, O_l and O_r. Epipolar line is the intersection of the epipolar plane with the image plane. The camera frames are related by a translation vector $T = (O_r - O_l)$ and a rotation matrix R. The relation between P_l and P_r (projection of P in the left and right frames) is given by $P_r = R(P_l - T)$. The usual equations of perspective projection define the relation between 3D points and their projections [57, 69]:

$$p_l = \frac{f_l}{Z_l} P_l \qquad \text{and} \qquad p_r = \frac{f_r}{Z_r} P_r \tag{7.1}$$

We assume the two cameras are parallel so $f_l = f_r$. Assume parallel optical axes, and known camera parameters (i.e., calibrated cameras), we can triangulate via similar triangles (p_l, P, p_r) and (O_l, P, O_r):

$$\frac{T + x_l - x_r}{Z - f} = \frac{T}{Z} \qquad \text{or} \qquad Z = f\frac{T}{x_r - x_l} \tag{7.2}$$

Thus:
$$d = x_r - x_l \tag{7.3}$$

where T is the stereo baseline and d measures the difference in retinal position between two corresponding points.

7.3 Multiview Image Acquisition

To acquire multiview images, one possible approach is to capture a texture image and the corresponding 3-D geometry of the scene. The 3-D geometry can be acquired by recording the scene from several viewpoints. In practice, two points of view corresponding to a left and right camera are usually employed. By comparing differences between the two captured images, the depth (that corresponds to the 3-D geometry) can be estimated and represented in a so-called depth image. This depth image is represented by a gray-scale image: usually dark and bright pixels correspond to foreground and background distance, respectively. By using a texture image and a corresponding depth image, one can perform depth image based rendering.

7.4 Depth Image Based Rendering

The DIBR is a key technology in an advanced 3D television system. Traditional 3D TV system requires the transmission of two video streams, the left and right view, to construct 3D vision. Unlike the traditional method, the advanced 3D TV system proposed a novel technology DIBR to provide 3D vision. DIBR uses intermediate view and intermediate depth map to render left and right view. In this way, broadcast content providers only have to transmit the left view and gray level depth map of the intermediate view.

Once intermediate image and depth image is given, any nearby image can be synthesized by mapping pixel coordinates one by one according to its depth value. However, there is an essential problem in DIBR that occlusion holes appear after pixel to pixel mapping. Holes do appear due to sharp horizontal changes in depth image, thus the location and size of holes differ from frame to frame. One solution to this problem is using 3D image warping technique [63, 64]. 3D image warping maps intermediate view pixel by pixel to left or right views according to the pixel's depth value. In other words, 3D image warping transforms pixel locations according to their depth values. Figure 7.5 shows an example of the 3D image warping technique using the left frame and the corresponding depth map.

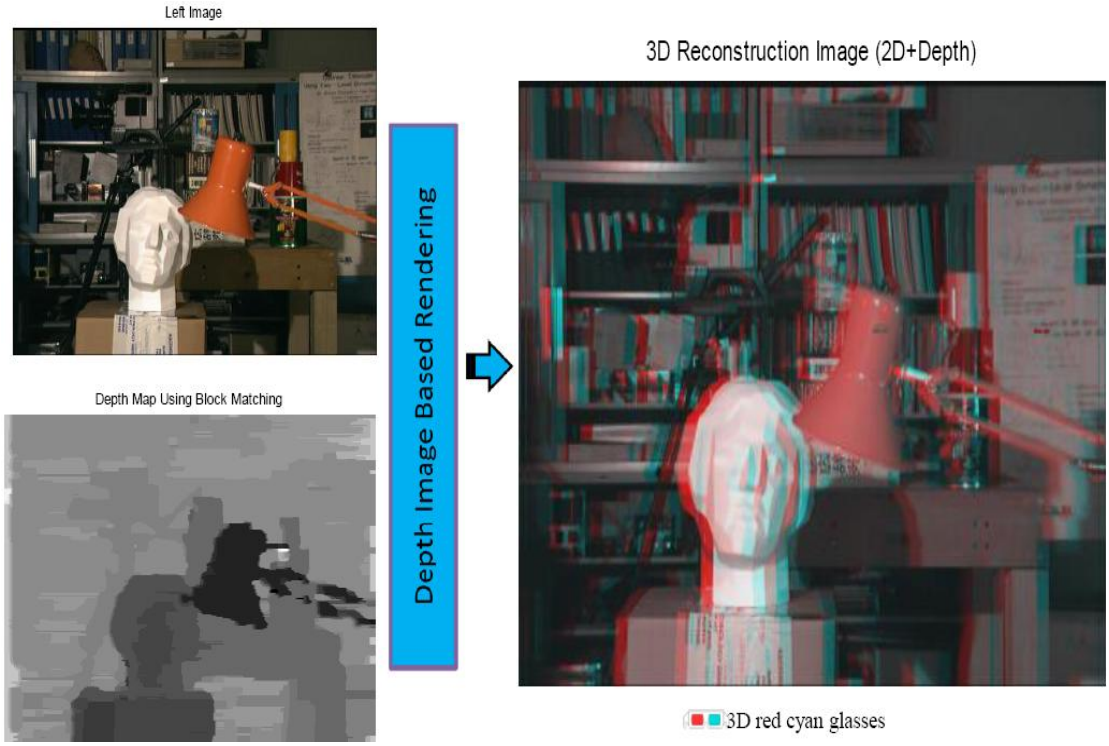


Figure 7.5 An example of the 3D image warping technique.

7.5 System Architecture for MB-VSBMC 3-D System

The encoder of our MB-VSBMC video-coding system is depicted in the block diagram in Figure 7.6. The depth frame is estimated using the left and right frame. The synchronized left texture frame and its corresponding depth frame are transformed into RDWT coefficients. Both ME and MC operations take place in the redundant wavelet domain for texture and depth images, as shown in the figure.

In the ME procedure, block matching is used to determine the motion of each set. Specifically, a block-matching procedure uses a cross-subband distortion that measures the sums of absolute differences for each block of the set. An adaptive variable size

window is used for the block search. The all-phase correlation edge mask and approximation subband (*LL*) are used to construct a multiband decision criteria for choosing the block size.

After the block size is determined, the motion from the reference frame to the current frame is estimated in the RDWT domain, and motion vectors are transmitted to the decoder. Multiband MC is accomplished by using a multiple reference frames (subbands) algorithm to generate a prediction frame. Residing in the RDWT domain, the motion-compensated residual is itself redundant; consequently, it is down-sampled before coding. The downsampling stage converts the overcomplete bands in RDWT to the critical DWT to be suitable for the encoding stage. The encoding step for CODEC consists of a set partitioning in hierarchical trees (SPIHT) algorithm for still image compression. We will use two synchronized encoders, one for the left view sequence and the other for its corresponding depth map.

The final step is DIBR which enables us to render the final frame to be ready for viewing by using the synchronized predicted left frame and its corresponding predicted depth frame.

As shown in Figure 7.6, the depth estimation process has been done in a spatial domain before the transformation in the redundant wavelet domain. Also in Figure 7.6, we divided the ME and MC blocks into two blocks, to indicate separate processes for the left texture frame and its corresponding depth frame.

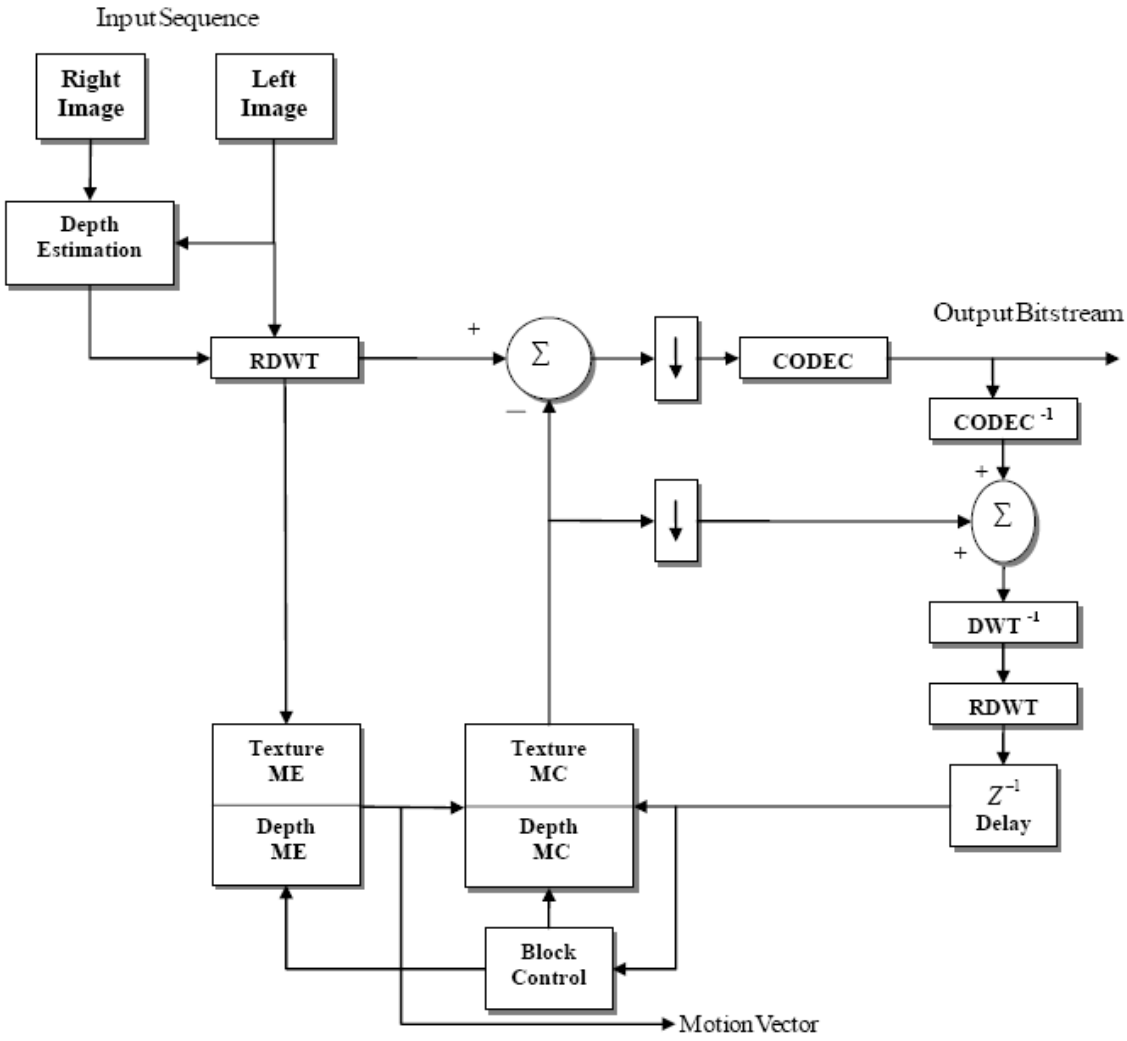


Figure 7.6 Block diagram of the MB-VSBMC 3D-video-coding system.

7.6 Experimental Results

In the experimental results we use the right and the left views for "Tiger" video, 320 frames of 352×288 pixels, with audio video interleave (AVI) DVD video format. The sequence is RGB24 and has a temporal sampling of 30 frame/ sec. For each synchronized right and left frame, we produced an estimated depth map. We perform ME/MC for each left frame and its estimated depth map separately. The first frame is intra-encoded (I-

frame) while all subsequent frames use ME/MC (P and B-frames). All wavelet transforms (RDWT) use the Daubechies 9-7 filter with symmetric extension and a decomposition of $J = 2$ level. The parameters α and β are 0.67 and 0.58 respectively. The core compression engine in all experiments is SPIHT. SPIHT produces an embedded coding rate with compression rate of 1 bpp. We used the 3D image warping technique to render and synthesize images, using a reference texture image and its corresponding depth image.

The depth image has a low energy and does not have sharp boundaries; therefore, it is not an easy task to obtain an accurate motion vector. Fortunately, the redundant wavelet domain provides a good solution by retaining all the phase information and providing a multiple prediction possibilities for motion techniques. Consequently, the proposed MB-VSBMC approach in a 3-D system may capture more motion contents of a depth map, and may result in better performance in terms of PSNR.

Figure 7.7 shows an example of an acquisition of 1-depth/1-texture. Figure 7.7.a and Figure 7.7.b are the 1^{st} and 11^{th} left-frame of the "Tiger" sequence, respectively. Figure 7.7.c and Figure 7.7.d are the corresponding depth map frames for the 1^{st} and 11^{th} frame, respectively. They are produced from the left and right texture frames using a depth estimation technique.

(a) 1st left-frame of the "Tiger" sequence.

(b) 11th left-frame of the "Tiger" sequence.

(c) The corresponding depth map for the 1st.

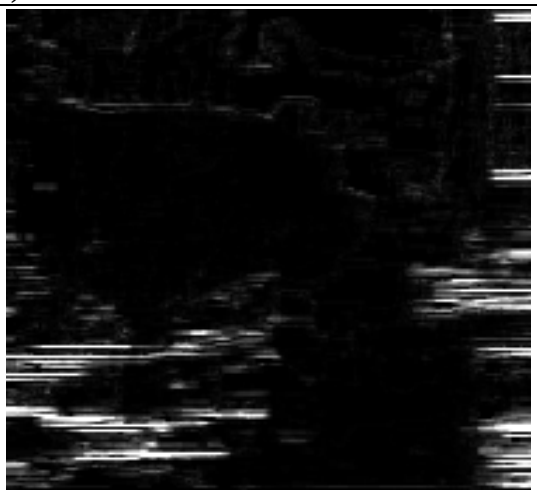
(d) The corresponding depth map for the 11th.

Figure 7.7 Example of an acquisition of 1-depth/1-texture.

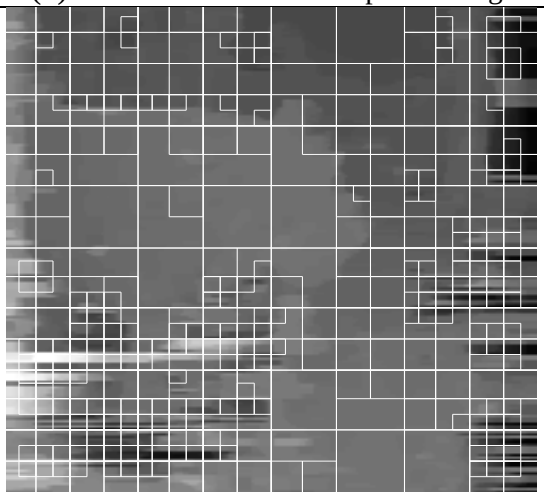
Figure 7.8 shows an example of MB-VSBMC block partitioning. Figure 7.8.a and Figure 7.8.c are the correlation edge masks of the 11th left-frame and its corresponding depth map frame of the "Tiger" sequence, respectively. Figure 7.8.b and Figure 7.8.d are the MB-VSBMC block partitionings for the 11th left-frame and its corresponding depth map frame, respectively.

(a) The correlation mask of the texture frame.

(b) The texture frame block partitioning.



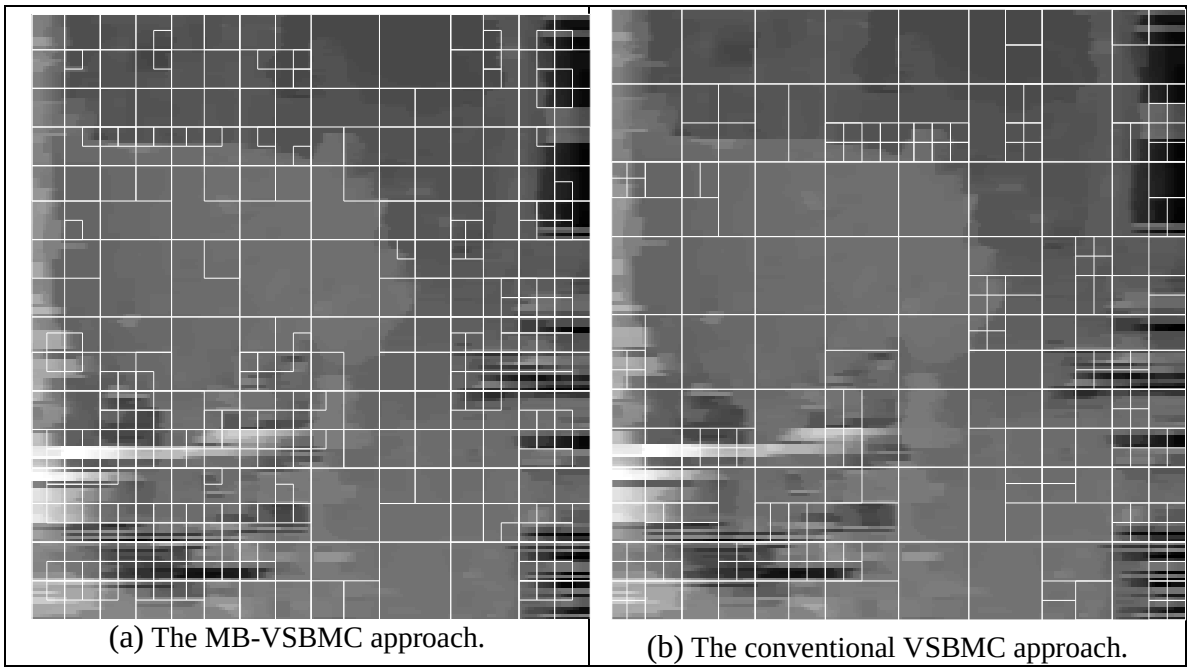
(c) The correlation mask of depth frame.



(d) The depth frame block partitioning.

Figure 7.8 Example of MB-VSBMC block partitionings.

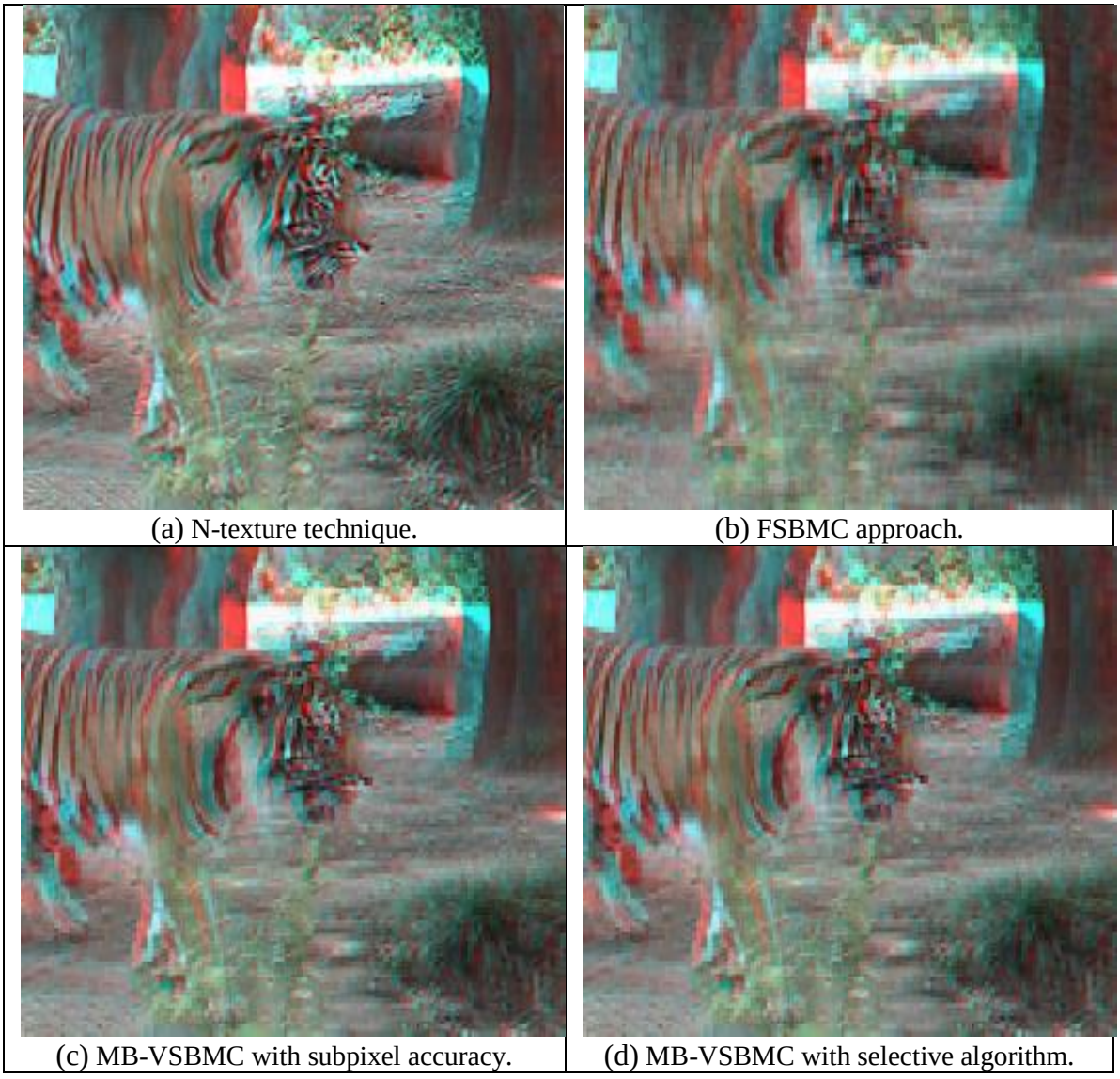
Figure 7.9 shows the comparison between two partitioning techniques for the depth map for the 11th frame of the "Tiger" sequence. Figure 7.9.a is the result of using a MB-VSBMC partitioning technique. Figure 7.9.b is the result of using a conventional VSBMC partitioning technique. Note that the MB-VSBMC approach shows superior performance for capturing the motion content.



(a) The MB-VSBMC approach. (b) The conventional VSBMC approach.

Figure 7.9 The comparison between two partitioning techniques for the depth map.

Figure 7.10 shows the 1-texture/ 1-depth comparison of the synthesized frame using three different block partitioning approaches in the redundant wavelet domain. Figure 7.10.a is the original synthesized 11^{th}-frame of the "Tiger" sequence from left and right frames using the N-texture technique. Figure 7.10.b is the synthesized frame produced using the FSBMC approach with subpixel accuracy. Figure 7.10.c is the synthesized frame produced using the MB-VSBMC approach with subpixel accuracy. Figure 7.10.d is the synthesized frame produced using the MB-VSBMC approach with selective algorithm.



(a) N-texture technique. (b) FSBMC approach.

(c) MB-VSBMC with subpixel accuracy. (d) MB-VSBMC with selective algorithm.

Figure 7.10 The 1-texture/ 1-depth comparison of the synthesized frames from different compression techniques.

Figure 7.11 shows the frame by frame PSNR comparisons for the "Tiger" sequence using the FSBMC with subpixel accuracy, the conventional VSBMC with subpixel accuracy, and the MB-VSBMC with either subpixel accuracy or selective algorithm. Table 7.1 shows an average PSNR comparison using the same techniques

mentioned above. The MB-VSBMC approach shows better performance than both the FSBMC and the conventional VSBMC in terms of PSNR. The results discussed here have included the OBMC approach.

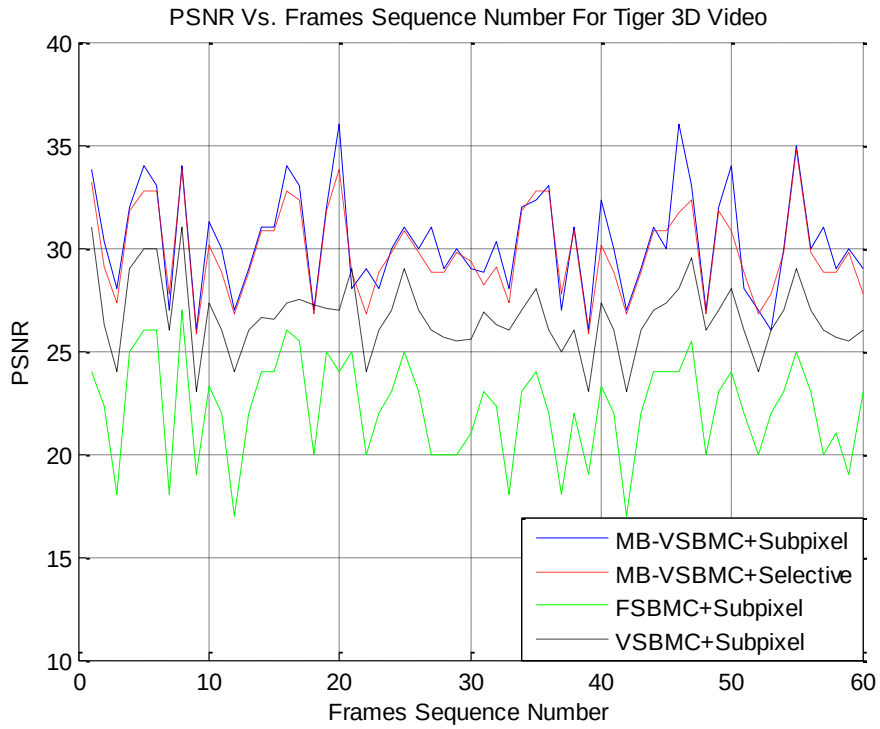


Figure 7.11 The frame by frame PSNR comparison with a CODEC bit rate of 1 bpp.

Table 7.1 An average PSNR comparison.

RDWT Domain	PSNR
FSBMC+Subpixel +OBMC	22.851
VSBMC+Subpixel+OBMC	27.329
MB-VSBMC+Subpixel+OBMC	30.011
MB-VSBMC+Selective +OBMC	29.567

CHAPTER 8

CONCLUSION

In this dissertation, we proposed a high performance video coding system based on the idea of ME/MC in the redundant wavelet domain. As demonstrated in a number of prior investigations in the RDWT domain, the shift variance of the usual critically sampled DWT no longer poses a problem for the estimation of object motion. However, as the research has demonstrated in this dissertation, the redundancy of RDWT can be exploited for advantages other than just its mere shift invariance. Specifically, the RDWT retains all the phases' information of a wavelet transform and facilitates the deployment of multiple-band evaluations for VSBMC.

The research presents a new adaptive partitioning scheme and decision criteria that utilizes more effectively the motion content of a frame in terms of the various block sizes. The new decision criterion partitions a given frame into variable size regions according to the motion information of the frame. The partitioning information is efficiently represented by a two-bit quadtree coding scheme. The frame partitioning is accomplished by first splitting and then merging processes.

In addition, the research investigates the rate allocation theory in the redundant wavelet domain to optimize the selection process of the block size. In view of the fact that an optimal partitioning scheme should minimize the coding rate as well as the prediction error of a frame, a choice between different MVs or different block sizes is

equivalent to a choice between points in the Rate-Distortion plane. This can be achieved by using the Lagrange multiplier method and solving the unconstrained problem.

The dissertation also proposed a selective subpixel accuracy algorithm for estimating the motion vector with a multiband approach. The selective subpixel approach reduces the computations produced by the conventional subpixel approach while maintaining almost the same accuracy. To enhance the quality of the system, the research applies the OBMC approach to mitigate the effects of blocking artifacts caused by the discontinuity among consecutive blocks in the motion-compensated frame.

In view of the fact that the 3D technology has been one of the fastest growing technologies in the recent years, the research extends the applications of the proposed MB-VSBMC to the 3D stereoscopic video coding system. The research approach is based on the structure of 1-texture/1-depth techniques and has employed the depth-based rendering to reconstruct the desired stereo views for each video frame.

Finally, the MB-VSBMC in redundant wavelet domain proposed in this dissertation follows the fact that the modern video systems are built upon a large collection of diverse techniques, all of which improve the system performance to various degrees. On future trend is to study the effectiveness of the algorithm using a content-driven rate-quality approach. In this case, the mean square error approach is no longer a valid criterion or measure of quality. The focus will change from rate distortion to rate quality optimization. It will require new quality assessment metrics and artifact detection methods related to the human perceptual responses.

REFERENCES

[1] J. R. Jain and A. K. Jain, "*Displacement measurement and its application interframe image coding,*" IEEE Transactions on Communications, vol. 29, no. 12, pp.1799–1808, December 1981.

[2] ITU-T, *Video Coding for Low Bitrate Communication*, November 1995, ITU-T Recommendation H.263, Version 1.

[3] ITU-T, *Video Coding for Low Bitrate Communication*, January 1998, ITU-T Recommendation H.263, Version 2.

[4] ISO/IEC 14496-2, *Information Technology—Coding of Audio-Visual Objects— Part 2: Visual*, 1999, MPEG-4 Coding Standard.

[5] H. W. Park and H. S. Kim, "*Motion estimation using low-band-shift method for wavelet-based moving-picture coding*," IEEE Transactions on Image Processing, vol. 9, no. 4, pp. 577–587, April 2000.

[6] H. S. Kim and H. W. Park, "*Wavelet-based moving-picture coding using shift invariant motion estimation in wavelet domain*," Signal Processing: Image Communication, vol. 16, no. 7, pp. 669–679, April 2001.

[7] X. Li, L. Kerofsky, and S. Lei, "*All-phase motion compensated prediction in the wavelet domain for high performance video coding*," in Proceedings of the International Conference on Image Processing, Thessaloniki, Greece, October 2001, vol. 2, pp. 538–541.

[8] X. Li and L. Kerofsky, "*High-performance resolution-scalable video coding via all-phase motion-compensated prediction of wavelet coefficients*," in Visual Communications and Image Processing, C.-C. J. Kuo, Ed. Proc. SPIE 4671, January 2002, pp. 1080–1090.

[9] X. Li and S. Lei, "*Efficient motion field representation in the wavelet domain for video compression*," in Proceedings of the International Conference on Image Processing, Rochester, NY, September 2002, vol. 3, pp. 257–260.

[10] G. Van der Auwera, A. Munteanu, P. Schelkens, and J. Cornelius, "*Scalable wavelet video-coding with in-band prediction—The bottom-up overcomplete discrete wavelet transform*," in Proceedings of the International Conference on Image Processing, Rochester, NY, September 2002, vol. 3, pp. 725–728.

[11] James E. Fowler, "*Analysis of Redundant-Wavelet Multihypothesis for Motion Compensation*," In Proceedings of the IEEE Data Compression Conference, J. A. Storer and M. Cohn, Eds., Snowbird, UT, March 2006, pp. 352.361.

[12] Suxia Cui and Yonghui Wang, "*Redundant Wavelet Transform in Video Signal Processing*," International Conference on Image Processing, Computer Vision, & Pattern Recognition (IPCV.06).

[13] P. Dutilleux, "An implementation of the "algorithme `a trous" to compute the wavelet transform," in *Wavelets: Time-FrequencyMethods and Phase Space*, J.-M.Combes, A. Grossman, and P. Tchamichian,Eds., pp. 298–304. Springer Verlag, Berlin, Germany, 1989, Proceedings of the International Conference, Marseille, France, December 14–18, 1987.

[14] S. Cui, Y. Wang, and J. E. Fowler, "*Multihypothesis motion compensation in the redundant wavelet domain*," In Proceedings of the International Conference on Image Processing, Barcelona, Spain, September 2003, vol. 2, pp. 53.56.

[15] James E. Fowler, Suxia Cui and Yonghui Wang, "*Motion Compensation Via Redundant- Wavelet Multihypothesis*," IEEE Transactions on Image Processing, October 2006 vol. 15, pp. 3102-3113.

[16] T. Koga, K. Iinuma, A. Hirano, Y. Iijima and T. Ishiguro, "*Motion compensated interframe coding for video conferencing*," in Proc. Nat. Telecommun. Conf., New Orleans, LA, 1981, pp .G5.3.1-G5.3.5.

[17] J. Jain and A. Jain, "*Displacement measurement and its application in interframe image coding*," Communications, IEEE Transactions, vol. 29, pp. 1799-1808, 1981.

[18] L. Reoxiang, Z. Bing, and M. L. Liou, "*A new three-step search algorithm for block motion estimation*," Circuits and Systems for Video Technology, *IEEE* ransactions, vol. 4, pp. 438-442, 1994.

[19] P. Lai-Man and M. Wing-Chung, "*A novel four-step search algorithm for fast block motion estimation*," Circuits and Systems for Video Technology, IEEE Transactions, vol. 6, pp. 313-317, 1996.

[20] K. Jong-Nam, B. Sung-Cheal, and A. Byung-Ha, "*Fast full search motion estimation algorithm using various matching scans in video coding*," Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions, vol. 31, pp. 540-548, 2001.

[21] Y. Wang, J. Ostermann, and Y. Q. Zhang, *Video Processing and Communications*: Prentice Hall, 2002.

[22] R. Injong, G. Martin, S. Muthukrishnan, and R. Packwood, "*Quad-tree structured variable-size block-matching motion estimation with minimal error*," Circuits and Systems for Video Technology, IEEE Transactions, vol. 10, pp. 42-50, 2000.

[23] G. R. Martin, R. A. Packwood, and I. Rhee, "*Variable size block matching motion estimation with minimal error*," in Proceedings of SPIE, Digital Video Compression: Algorithms and Technologies 1996, pp. 324-333.

[24] K.H. Lee, J.H. Choi, B.K. Lee, and D.G. Kim "*Fast Two-Step Half-Pixel Accuracy Motion Vector Prediction*," Electronics Letters, vol. 36, pp. 625-627, Mar. 2000.

[25] X. Li and C. Gonzales, "*A Locally Quadratic Model of the Motion Estimation Error Criterion Function and Its Application to Subpixel Interpolation*," IEEE Trans. Circuts Syst. Video Technol., vol. 6, pp. 118-122, Feb. 1996.

[26] B. Girod, "*Motion-compensating prediction with fractional-peel accuracy*", IEEE Trans. Comm., vol. 41, no. 4, pp. 604–612, Apr. 1993.

[27] 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, June 1996.

[28] Gary J. Sullivan and Richard L. Baker, "*Rate – Distortion Optimized Motion Compensation For Video Compression Using Fixed or Variable Size Blocks*," Global Telecommunications Conference, 1991, pp. 85-90.

[29] Y. Shoham and A. Gersho, "*Efficient codebook allocation for an arbitrary set of vector quantizers*," in IEEE Int. Conf. on Acoust., Speech, Signal Processing (ICASSP), pp. 43.7.1-4, 1985.

[30] Y. Shoham and A. Gersho, "*Efficient bit allocation for an arbitrary set of quantizes*," IEEE Trans. Acoust., Speech, Signal Processing, vol. ASSP-36, pp. 1445-1453, Sept. 1988.

[31] H. Everett III, "*Generalized Lagrange multiplier method for solving problems of optimum allocation of resources*," Operations Research, vol. 11, pp. 399- 417, 1963.

[32] M. H. Chan, Y. B. Yu, and A. G. Constantinides, "*Variable size block matching motion compensation with applications to video coding*," IEE Proceedings, vol. 137, Part 1, pp. 205-212, Aug. 1990.

[33] G. J. Sullivan and R. L. Baker, "Efficient *quadtree coding of images and video*," in IEEE Int. Conf. on Acoust., Speech, Signal Processing (ICASSP), pp. 2661-2664, May 1991.

[34] D. J. Vaisey and A. Gersho, "*Variable block-size image coding*," IEEE Int. Conf. on Acoust., Speech, Signal Processing (ICASSP), pp. 25.1.1-4, Apr. 1987.

[35] P. Strobach, "*Tree-structured scene adaptive coder*," IEEE Trans. Commun., vol. COM-38, pp. 477-486, Apr. 1990.

[36] J. Zhang, M. Omair and M. Swamy, "*New windowing techniques for variable-size block motion compensation*," IEEE Proc.Vis. Image Signal Processing, vol. 145, No. 6, December 1998.

[37] S. Cui, Y. Wang, and J. E. Fowler, "*Mesh-based motion estimation and compensation in the wavelet domain using a redundant transform*," in Proceedings of the International Conference on Image Processing, Rochester, NY, September 2002, vol. 1, pp. 693–696.

[38] Jooyoung Jung, et al. "*Fast Subpel Motion Estimation Using Selective Motion Vector Accuracy of Inter-Mode Decision for H.264/AVC*," IEEE International Symposium on Industrial Electronics (ISlE 2009) Seoul Olympic Parktel, Seoul, Korea July 5-8, 2009.

[39] Z. Wang, A. C. Bovik, H. R. Sheikh and E. P. Simoncelli, "*Image quality assessment: From error visibility to structural similarity*," IEEE Transactions on Image Processing 13, 600-612 (2004).

[40] J. Lee, "*Optimal quadtree for variable block size motion estimation*," ICIP.96, 1996, vol.3, pp. 480 .483.

[41] S. Nogaki and M. Ohta, "*An overlapped block motion compensation for high quality motion picture coding*," in Proceedings of the IEEE International Symposium on Circuits and Systems, San Diego, CA, May 1992, vol. 1, pp. 184–187.

[42] M. T. Orchard and G. J. Sullivan, "*Overlapped block motion compensation: An estimation-theoretic approach*," IEEE Transactions on Image Processing, vol. 3, no. 5, pp. 693–699, September 1994.

[43] J. K. Su and R. M. Mersereau, "*Motion estimation methods for overlapped block motion compensation*," IEEE Trans. Image Processing, vol. 9, pp. 1509–1521, 2000.

[44] W. Zheng, Y. Shishikui, and M. Naemura, "*Analysis of overlapped block motion compensation based on a statistical motion distribution model*," Proceedings of the International Conference on Image Processing, vol. 3, pp. 522–525, 2001.

[45] R. Rajagopalan, E. Feig, T. Orchard and J. Watson, "*Motion Optimization of Ordered Blocks for Overlapped Block Motion Compensation*," Circuits and Systems for Video Technology, IEEE Transactions on Apr 1998, pp. 119 – 123.

[46] Gary J. Sullivan and Michael T. Orchard, "*Methods of reduced-complexity overlapped block motion compensation*," Proceedings. ICIP-94., IEEE International Conference, vol.2, pp. 957-961.

[47] J. Chen, J. Xu, D. Xiang and C. Geng, "*Implementation of Multiple Macroblock Mode Overlapped Block Motion Compensation for Wavelet Video Coding*," Circuits Systems Signal Processing, vol. 26, No. 1, 2007, pp. 55–67.

[48] H. Watanabe and S. Singhal, "*Windowed Motion Compensation*," In Proc. of SPIE Conf. on Visual Commun. and Image Proc., vol. 1605, Pt. 2, pp. 582-589, Nov. 1991.

[49] J. Zhang, M. Omair, and M. Swamy," *A New Variable Size Block Motion Compensation*," In Proceedings of the IEEE1997, pp. 164-167.

[50] J. Zhang, M. Omair, and M. Swamy, "*New windowing techniques for variable-size block motion compensation*," IEEE Proc.Vis. Image Signal Process., vol. 145, No. 6, December 1998.

[51] W. Matusik and H. Pfister, "*3D TV: a scalable system for real-time acquisition, transmission, and autostereoscopic display of dynamic scenes*," ACM Transactions on Graphics, vol. 23, no. 3, pp. 814–824, 2004.

[52] A. Vetro, P. Pandit, H. Kimata, A. Smolic, and Y.-K.Wang, "*Joint draft 8.0 on multiview video coding*." Joint Video Team (JVT) of ISO/IEC MPEG ITU-T VCEG ISO/IEC JTC1/SC29/WG11 and ITU-T SG16 Q.6, July 2008.

[53] U. Fecker and A. Kaup, "*H.264/AVC compatible coding of dynamic light fields using transposed picture ordering*," in Proceedings of the European Signal Processing Conference (EUSIPCO), vol. 1, (Antalya, Turkey), September 2005.

[54] P. Merkle, K. Mueller, A. Smolic, and T.Wiegand, "*Efficient compression of multi-view video exploiting inter-view dependencies based on H.264/MPEG4-AVC*," in IEEE International Conference on Multimedia and Expo, (Toronto, Canada), pp. 1717–1720, July 2006.

[55] J. H. Kim, P. Lai, J. Lopez, A. Ortega, Y. Su, P. Yin, and C. Gomila, "*New coding tools for illumination and focus mismatch compensation in multiview video coding*," IEEE Transactions on Circuits and Systems for Video Technology, vol. 17, No. 11, pp. 1519–1535, 2007.

[56] Yannick Morvan, Dirk Farin and Peter H.N,"*System Architecture for Free-Viewpoint Video and 3D-TV*," IEEE Transactions on Consumer Electronics, Vol. 54, No. 2, May 2008.

[57] C. Fehn, "*Depth-image-based rendering (DIBR), compression, and transmission for a new approach on 3D-TV*," in Proceedings of the SPIE, Stereoscopic Displays and Virtual Reality Systems XI, vol. 5291, pp. 93–104, 2004.

[58] A. Bourge and C. Fehn, "*White paper on ISO/IEC 23002-3 auxiliary video data representations*," ISO/IEC JTC1/SC29/WG11/N8039, April 2006.

[59] Y. Luo, Z. Zhang, and P. An, "*Stereo video coding based on frame estimation and interpolation*," IEEE Trans. on Broadcast. 49, pp. 14.21, 2003.

[60] Y. Morvan, D. Farin, and P. H. N. deWith, "*Design considerations for a 3D-TV video coding architecture*," in IEEE International Conference on Consumer Electronics, January 2008.

[61] Y. Morvan, "*Acquisition, compression and rendering of depth and texture for multi-view video*", PhD thesis, Eindhoven University of Technology, 2009.

[62] Applications and Requirements for Stereo-scopic Video (SSV). ISO/IEC JTC1/ SC29/ WG11 Bangkok, Thailand, January 2006.

[63] Y. Morvan, P. H. N. de With, and D. Farin, "*Platelet-based coding of depth maps for the transmission of multiview images*," in Proceedings of the SPIE, Stereoscopic Displays and Virtual Reality Systems XIII, vol. 6055, (San Jose, USA), p. 60550K, January 2006.

[64] L. McMillan, "*An Image-Based Approach to Three-Dimensional Computer Graphics,"* PhD thesis, University of North Carolina, Chapel Hill, USA, April 1997.

[65] S. Cui, "*Motion Estimation and Compensation in the Redundant Wavelet Domain*," PhD thesis, Mississippi State University, 2003.

[66] Kannan Ramchandran and Martin Vetterli, "*Best Wavelet Packet Bases in a Rate-*

Distortion Sense," IEEE Trans. On image processing. Vol. 2, NO. 2, pp. 160-175, April 1993.

[67] H. Watanabe and S. Singhal. "*Windowed motion compensation*," In Visual Communications and Image Processing 91, Vol. 1605, pp. 582-589, 1991.

[68] Y. Morvan, D. Farin, P. H. N. de With, "*Depth-Image Compression based on an R-D Optimized Quadtree Decomposition for the Transmission of Multiview Images*," in Proceedings of the IEEE International Conference on Image Processing (ICIP), San Antonio, TX, USA, September 2007, pp. V-105–108.

[69] Y. Mori, N. Fukushima, T. Fujii, and M. Tanimoto, "*View Generation with 3D Warping Using Depth Information for FTV*," in Proceedings of 3DTV-Conference, pp. 229-232, 2008.

APPENDIX

MATLAB CODE

```
% Main code for 2D MB-VSBMC in RDWT
% Ahmed Suliman 23 Feb 2010
clear all
close all
clc
tic

% Create a new AVI file to store the output AVI file
aviobj = avifile('test_out.avi','fps',25,'COMPRESSION','None');
% Initialize motion Vectors for splitting process
motionVect=zeros(2,256);
motionVect1=zeros(2,1024);
motionVect2=zeros(2,4096);
%flag to tell if B frame was predicted from I or P frame
%flag=1 P frame & flag=0 I frame & flag=2 both I & P
flag=0;
% Collect info about input file
mov=aviread('test.avi');
movinfo=aviinfo('test.avi');
noframe=movinfo.NumFrames;
% 3 Steps ME algo initialization
% mbSize indicate the Max MB size used for ME (splitting process)
mbSize = 16;
%p for search area
p = 7;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
global I1d_LH P4d_LH P7d_LH I10d_LH B2d_LH B3d_LH B5d_LH B6d_LH B8d_LH
B9d_LH
global I1d_HH P4d_HH P7d_HH I10d_HH B2d_HH B3d_HH B5d_HH B6d_HH B8d_HH
B9d_HH
global I1d_HL P4d_HL P7d_HL I10d_HL B2d_HL B3d_HL B5d_HL B6d_HL B8d_HL
B9d_HL
global I1d P4d P7d I10d B2d B3d B5d B6d B8d B9d
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Extract I frame
framedata=aviread('test.avi',1);
I1=frame2im(framedata);
I_1=imresize(I1,[256 256]);
I1=rgb2gray(I_1);
I1=double(I1);
% Transfer I frame into RDWT
h = daubcqf(6);
[ll_lev2,yh,L] = mrdwt(I1,h,1);
N = 256;
lh = yh(:,1:N);
```

```
hl = yh(:,N+1:2*N);
hh = yh(:,2*N+1:3*N);
% Store the feedback buffer frame
JQ=ll_lev2;
bufferI1= mirdwt(JQ,yh,h,1);
c= makeLayers(JQ);
% Calling decoder
identifier=1;
decoder1(c,motionVect,identifier,flag);
identifier=1;
decoder_I(c,1,motionVect,identifier,flag);
% Calculat frame by frame PSNR
ESpsnr(1) = imgPSNR(I1_Dec, frame_1, 255);
%++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
% Group frames intio I, P and B
% Initialization
 k=1;
% Extracting frames from input
for i=1:10:10
    i
    if i~=1
        j=i-round(i/10);
        [B2,B3,P4,B5,B6,P7,B8,B9,I10]=GOP(j);
    elseif i==1
        j=i;
        [B2,B3,P4,B5,B6,P7,B8,B9,I10]=GOP(j);
    end

% Start to predict P4 from I1
% Obtain all phase correlation mask
Mask_R=Corr_Mask(I1);
Mask_t=Corr_Mask(P4);
% ME/MC for all bands in frame 4
[bufferP4,streamP4,motionVect]=compensatedFrame_Mod(P4,I1,Mask_R,mbSize
,p);
c4= makeLayers(streamP4);
%calling decoder
identifier=4;
%decoder
decoder1(c4,motionVect,identifier,flag);
[bufferP4_LH,streamP4_LH,motionVect,motionVect14]=compensatedFrame_LH_M
od(P4,I1,Mask_R,Mask_t,mbSize,p,motionVect,1);
c_LH= makeLayers(streamP4_LH);
%calling decoder
identifier=4;
%decoder
decoder_LH(c_LH,1,motionVect,identifier,flag);
[bufferP4_HL,streamP4_HL,motionVect]=compensatedFrame_HL_Mod(P4,I1,Mask
_R,Mask_t,mbSize,p,motionVect,1);
c_HL= makeLayers(streamP4_HL);
%calling decoder
identifier=4;
%decoder
```

```
decoder_HL(c_HL,1,motionVect,identifier,flag);
[bufferP4_HH,streamP4_HH,motionVect]=compensatedFrame_HH_Mod(P4,I1,Mask
_R,Mask_t,mbSize,p,motionVect,1);
c_HH= makeLayers(streamP4_HH);
%calling decoder
identifier=4;
%decoder
decoder_HH(c_HH,1,motionVect,identifier,flag);
% Retrived reconstructed frame
yh1=[P4d_LH,P4d_HL,P4d_HH];
frame_4= mirdwt(P4d,yh1,h,1);
% Calculate frame by frame PSNR
ESpsnr(k+3) = imgPSNR(P4, frame_4, 255);

% ME/MC for all bands in frame 2
[B2t ,motionVect,motionVect1] =
bFrameProc_Mod(B2,bufferI1,bufferP4,Mask_R,Mask_t,mbSize,p);
c= makeLayers(B2t);
%calling decoder
flag=2;
identifier=2;
decoder1(c,motionVect,identifier,flag);
[B2t_LH ,motionVect,motionVect1, flag] =
bFrameProc1_LH_Mod(B2,bufferI1,bufferP4_LH,Mask_R,Mask_t,mbSize,p);
c_LH= makeLayers(B2t_LH);
%calling decoder
identifier=2;
%decoder
decoder_LH(c_LH,1,motionVect,identifier,flag);
[B2t_HL ,motionVect,motionVect1, flag] =
bFrameProc1_HL_Mod(B2,bufferI1,bufferP4_HL,Mask_R,Mask_t,mbSize,p);
c_HL= makeLayers(B2t_HL);
identifier=2;
%decoder
decoder_HL(c_HL,1,motionVect,identifier,flag);
[B2t_HH ,motionVect,motionVect1, flag] =
bFrameProc1_HH_Mod(B2,bufferI1,bufferP4_HH,Mask_R,Mask_t,mbSize,p);
c_HH= makeLayers(B2t_HH);
%calling decoder
identifier=2;
%decoder
decoder_HH(c_HH,1,motionVect,identifier,flag);
% Retrived reconstructed frame
yh2=[B2d_LH,B2d_HL,B2d_HH];
frame_2= mirdwt(B2d,yh2,h,1);
% Calculate frame by frame PSNR
ESpsnr(k+1) = imgPSNR(B2, frame_2, 255);

% ME/MC for all bands in frame 3
[B3t ,motionVect,motionVect1] =
bFrameProc_Mod(B3,bufferI1,bufferP4,Mask_R,Mask_t,mbSize,p);
c= makeLayers(B3t);
%calling decoder
```

```
identifier=3;
decoder1(c,motionVect,identifier,flag);
[B3t_LH ,motionVect,motionVect1, flag] =
bFrameProc1_LH_Mod(B3,bufferI1,bufferP4_LH,Mask_R,Mask_t,mbSize,p);
c_LH= makeLayers(B3t_LH);
%calling decoder
identifier=3;
%decoder
decoder_LH(c_LH,1,motionVect,identifier,flag);
[B3t_HL ,motionVect,motionVect1, flag] =
bFrameProc1_HL_Mod(B3,bufferI1,bufferP4_HL,Mask_R,Mask_t,mbSize,p);
c_HL= makeLayers(B3t_HL);
%calling decoder
identifier=3;
%decoder
decoder_HL(c_HL,1,motionVect,identifier,flag);
[B3t_HH ,motionVect,motionVect1, flag] =
bFrameProc1_HH_Mod(B3,bufferI1,bufferP4_HH,Mask_R,Mask_t,mbSize,p);
c_HH= makeLayers(B3t_HH);
%calling decoder
identifier=3;
%decoder
decoder_HH(c_HH,1,motionVect,identifier,flag);
% Retrived reconstructed frame
yh3=[B3d_LH,B3d_HL,B3d_HH];
frame_3= mirdwt(B3d,yh3,h,1);
% Calculate frame by frame PSNR
ESpsnr(k+2) = imgPSNR(B3, frame_3, 255);

% ME/MC for all bands in frame 7
% Predicting p7 from p4
[bufferP7,streamP7,motionVect]=compensatedFrame_Mod(P7,P4,Mask_t,mbSize
,p);
c= makeLayers(streamP7);
%calling decoder
identifier=7;
decoder1(c,motionVect,identifier,flag);
[bufferP7_LH,streamP7_LH,motionVect,motionVect17]=compensatedFrame_LH_M
od(P7,P4,Mask_t,Mask_t2,mbSize,p,motionVect,1);
c_LH= makeLayers(streamP7_LH);
%calling decoder
identifier=7;
%decoder
decoder_LH(c_LH,1,motionVect,identifier,flag);
[bufferP7_HL,streamP7_HL,motionVect]=compensatedFrame_HL_Mod(P7,P4,Mask
_t,Mask_t2,mbSize,p,motionVect,1);
c_HL= makeLayers(streamP7_HL);
%calling decoder
identifier=7;
%decoder
decoder_HL(c_HL,1,motionVect,identifier,flag);
[bufferP7_HH,streamP7_HH,motionVect]=compensatedFrame_HH_Mod(P7,P4,Mask
_t,Mask_t2,mbSize,p,motionVect,1);
```

```
c_HH= makeLayers(streamP7_HH);
%calling decoder
identifier=7;
%decoder
decoder_HH(c_HH,1,motionVect,identifier,flag);
% Retrived reconstructed frame
yh7=[P7d_LH,P7d_HL,P7d_HH];
frame_7= mirdwt(P7d,yh7,h,1);
% Calculate frame by frame PSNR
ESpsnr(k+6) = imgPSNR(P7, frame_7, 255);

% ME/MC for all bands in frame 5
[B5t, motionVect ,motionVect1] =
bFrameProc_Mod(B5,P4,bufferP7,Mask_t,Mask_t2,mbSize,p);
c= makeLayers(B5t);
flag=0;
%calling decoder
identifier=5;
decoder1(c,motionVect,identifier,flag);
[B5t_LH ,motionVect,motionVect1, flag] =
bFrameProc1_LH_Mod(B5,P4,bufferP7_LH,Mask_t,Mask_t2,mbSize,p);
c_LH= makeLayers(B5t_LH);
%calling decoder
identifier=5;
%decoder
decoder_LH(c_LH,1,motionVect,identifier,flag);
[B5t_HL ,motionVect,motionVect1, flag] =
bFrameProc1_HL_Mod(B5,P4,bufferP7_HL,Mask_t,Mask_t2,mbSize,p);
c_HL= makeLayers(B5t_HL);
%calling decoder
identifier=5;
%decoder
decoder_HL(c_HL,1,motionVect,identifier,flag);
[B5t_HH ,motionVect,motionVect1, flag] =
bFrameProc1_HH_Mod(B5,P4,bufferP7_HH,Mask_t,Mask_t2,mbSize,p);
c_HH= makeLayers(B5t_HH);
%calling decoder
identifier=5;
%decoder
decoder_HH(c_HH,1,motionVect,identifier,flag);
% Retrived reconstructed frame
yh5=[B5d_LH,B5d_HL,B5d_HH];
frame_5= mirdwt(B5d,yh5,h,1);
% Calculate frame by frame PSNR
ESpsnr(k+4) = imgPSNR(B5, frame_5, 255);

% ME/MC for all bands in frame 6
[B6t, motionVect,motionVect1] =
bFrameProc_Mod(B6,P4,bufferP7,Mask_t,Mask_t2,mbSize,p);
c= makeLayers(B6t);
%calling decoder
identifier=6;
decoder1(c,motionVect,identifier,flag);
```

```
[B6t_LH ,motionVect,motionVect1, flag] =
bFrameProc1_LH_Mod(B6,P4,bufferP7_LH,Mask_t,Mask_t2,mbSize,p);
c_LH= makeLayers(B6t_LH);
%calling decoder
identifier=6;
%decoder
decoder_LH(c_LH,1,motionVect,identifier,flag);
[B6t_HL ,motionVect,motionVect1, flag] =
bFrameProc1_HL_Mod(B6,P4,bufferP7_HL,Mask_t,Mask_t2,mbSize,p);
c_HL= makeLayers(B6t_HL);
%calling decoder
identifier=6;
%decoder
decoder_HL(c_HL,1,motionVect,identifier,flag);
[B6t_HH ,motionVect,motionVect1, flag] =
bFrameProc1_HH_Mod(B6,P4,bufferP7_HH,Mask_t,Mask_t2,mbSize,p);
c_HH= makeLayers(B6t_HH);
%calling decoder
identifier=6;
%decoder
decoder_HH(c_HH,1,motionVect,identifier,flag);
% Retrived reconstructed frame
yh6=[B6d_LH,B6d_HL,B6d_HH];
frame_6= mirdwt(B6d,yh6,h,1);
% Calculate frame by frame PSNR
ESpsnr(k+5) = imgPSNR(B6, frame_6, 255);

% Processing on I10 frame
% Transfer I frame into RDWT
h = daubcqf(6);
[ll_lev221,yh2,L] = mrdwt(I10,h,1);
N = 256;
lh_le = yh2(:,1:N);
hl_le = yh2(:,N+1:2*N);
hh_le = yh2(:,2*N+1:3*N);
streamI10=ll_lev221;
bufferI10=mirdwt(streamI10,yh2,h,1);
% Obtain all phase correlation mask
Mask_le=Corr_Mask(I10);
c= makeLayers(streamI10);
%calling decoder
identifier=10;
decoder1(c,motionVect,identifier,flag);
identifier=10;
decoder_LH(lh_le,1,motionVect,identifier,flag);
decoder_HL(hl_le,1,motionVect,identifier,flag);
decoder_HH(hh_le,1,motionVect,identifier,flag);
% Retrived reconstructed frame
yh10=[I10d_LH,I10d_HL,I10d_HH];
frame_10= mirdwt(I10d,yh10,h,1);
% Calculate frame by frame PSNR
ESpsnr(k+9) = imgPSNR(I10, frame_10, 255);
```

```
% ME/MC for all bands in frame 8
[B8t, motionVect, motionVect1] =
bFrameProc_Mod(B8,P7,streamI10,Mask_le,Mask_t2,mbSize,p);
c= makeLayers(B8t);
flag=2;
%calling decoder
identifier=8;
decoder1(c,motionVect,identifier,flag);
[B8t_LH ,motionVect, motionVect1, flag] =
bFrameProc1_LH_Mod(B8,P7,I10d_LH,Mask_le,Mask_t2,mbSize,p);
c_LH= makeLayers(B8t_LH);
%calling decoder
identifier=8;
%decoder
decoder_LH(c_LH,1,motionVect,identifier,flag);
[B8t_HL ,motionVect, motionVect1, flag] =
bFrameProc1_HL_Mod(B8,P7,I10d_HL,Mask_le,Mask_t2,mbSize,p);
c_HL= makeLayers(B8t_HL);
%calling decoder
identifier=8;
%decoder
decoder_HL(c_HL,1,motionVect,identifier,flag);
[B8t_HH ,motionVect, motionVect1, flag] =
bFrameProc1_HH_Mod(B8,P7,I10d_HH,Mask_le,Mask_t2,mbSize,p);
c_HH= makeLayers(B8t_HH);
%calling decoder
identifier=8;
%decoder
decoder_HH(c_HH,1,motionVect,identifier,flag);
% Retrived reconstructed frame
yh8=[B8d_LH,B8d_HL,B8d_HH];
frame_8= mirdwt(B8d,yh8,h,1);
% Calculate frame by frame PSNR
ESpsnr(k+7) = imgPSNR(B8, frame_8, 255);

% ME/MC for all bands in frame 9
[B9t, motionVect,  motionVect1] =
bFrameProc_Mod(B9,P7,streamI10,Mask_le,Mask_t2,mbSize,p);
c= makeLayers(B9t);
%calling decoder
identifier=9;
decoder1(c,motionVect,identifier,flag);
[B9t_LH ,motionVect, motionVect1, flag] =
bFrameProc1_LH_Mod(B9,P7,I10d_LH,Mask_le,Mask_t2,mbSize,p);
c_LH= makeLayers(B9t_LH);
%calling decoder
identifier=9;
%decoder
decoder_LH(c_LH,1,motionVect,identifier,flag);
[B9t_HL ,motionVect, motionVect1, flag] =
bFrameProc1_HL_Mod(B9,P7,I10d_HL,Mask_le,Mask_t2,mbSize,p);
c_HL= makeLayers(B9t_HL);
%calling decoder
```

```
identifier=9;
%decoder
decoder_HL(c_HL,1,motionVect,identifier,flag);
[B9t_HH ,motionVect, motionVect1, flag] =
bFrameProc1_HH_Mod(B9,P7,I10d_HH,Mask_le,Mask_t2,mbSize,p);
c_HH= makeLayers(B9t_HH);
%calling decoder
identifier=9;
%decoder
decoder_HH(c_HH,1,motionVect,identifier,flag);
% Retrived reconstructed frame
yh9=[B9d_LH,B9d_HL,B9d_HH];
frame_9= mirdwt(B9d,yh9,h,1);
% Calculate frame by frame PSNR
ESpsnr(k+8) = imgPSNR(B9, frame_9, 255);
k=k+9;

% Reorder the frames to reconstruct the output sequence
if i==1
    imshow(uint8(frame_1),[])
    aviobj = addframe(aviobj,getframe);
    imshow(uint8(frame_2),[])
    aviobj = addframe(aviobj,getframe);
    imshow(uint8(frame_3),[])
    aviobj = addframe(aviobj,getframe);
    imshow(uint8(frame_4),[])
    aviobj = addframe(aviobj,getframe);
    imshow(uint8(frame_5),[])
    aviobj = addframe(aviobj,getframe);
    imshow(uint8(frame_6),[])
    aviobj = addframe(aviobj,getframe);
    imshow(uint8(frame_7),[])
    aviobj = addframe(aviobj,getframe);
    imshow(uint8(frame_8),[])
    aviobj = addframe(aviobj,getframe);
    imshow(uint8(frame_9),[])
    aviobj = addframe(aviobj,getframe);
    imshow(uint8(frame_10),[])
    aviobj = addframe(aviobj,getframe);
else
    imshow(uint8(frame_2),[])
    aviobj = addframe(aviobj,getframe);
    imshow(uint8(frame_3),[])
    aviobj = addframe(aviobj,getframe);
    imshow(uint8(frame_4),[])
    aviobj = addframe(aviobj,getframe);
    imshow(uint8(frame_5),[])
    aviobj = addframe(aviobj,getframe);
    imshow(uint8(frame_6),[])
    aviobj = addframe(aviobj,getframe);
    imshow(uint8(frame_7),[])
    aviobj = addframe(aviobj,getframe);
    imshow(uint8(frame_8),[])
```

```
    aviobj = addframe(aviobj,getframe);
    imshow(uint8(frame_9),[])
    aviobj = addframe(aviobj,getframe);
    imshow(uint8(frame_10),[])
    aviobj = addframe(aviobj,getframe);
end
I1=I10;
bufferI1=bufferI10;
I1d=I10d;
end
aviobj = close(aviobj);
toc


function Mask=Corr_Mask(I1)
% Construct the all phase correlation mask

h = daubcqf(6);
[ll_lev2,yh,L] = mrdwt(I1/max(max(I1)),h,1);
N = 256;
lh = yh(:,1:N);
hl = yh(:,N+1:2*N);
hh = yh(:,2*N+1:3*N);
lh_lev2 = yh(:,3*N+1:4*N);
hl_lev2 = yh(:,4*N+1:5*N);
hh_lev2 = yh(:,5*N+1:6*N);
Mask=abs(lh.*lh_lev2 )+abs(hl.*hl_lev2)+abs(hh.*hh_lev2);


function
[bufferImageP,streamP,motionVect]=compensatedFrame_Mod(Im,bufferImage,M
ask,mbSize,p)
% Function ME/MC LL-band

h = daubcqf(6);
[ll_lev2,yh,L] = mrdwt(Im,h,1);
Im=ll_lev2;
[ll_lev22,yh1,L1] = mrdwt(bufferImage,h,1);
bufferImage=ll_lev22;
% 3-steps ME
motionVect = motionEstTSS(Im,bufferImage,mbSize,p);
% MC
imgComp = motionComp(bufferImage, motionVect, mbSize);
% obtain Resduial
imageSubtract=imsubtract_W(abs(Im),abs(imgComp));
bufferImageP=(imadd(abs(Im),abs(imgComp)));
bufferImageP= mirdwt(bufferImageP,yh1,h,1);
streamP=imageSubtract;


function
[bufferImageP,streamP,motionVect,motionVect1]=compensatedFrame_LH_Mod(I
m,bufferImage,Mask,Mask_1,mbSize,p,motionVect,level)
```

```
% Function to ME/MC LH-band

h = daubcqf(6);
[ll_lev2,yh,L] = mrdwt(Im,h,1);
N = 256;
lh = yh(:,1:N);
hl = yh(:,N+1:2*N);
hh = yh(:,2*N+1:3*N);
lh_lev2 = yh(:,3*N+1:4*N);
hl_lev2 = yh(:,4*N+1:5*N);
hh_lev2 = yh(:,5*N+1:6*N);

if level==1
    Im=lh;
else
    Im=lh_lev2;
end

[ll_lev22,yh1,L1] = mrdwt(bufferImage,h,1);
lh1 = yh1(:,1:N);
hl1 = yh1(:,N+1:2*N);
hh1 = yh1(:,2*N+1:3*N);
lh_lev21 = yh1(:,3*N+1:4*N);
hl_lev21 = yh1(:,4*N+1:5*N);
hh_lev21 = yh1(:,5*N+1:6*N);

if level==1
    bufferImage =lh1;
else
    bufferImage =lh_lev21;
end

motionVect = motionEstTSS(Im,bufferImage,mbSize,p);
motionVect1 = motionEstTSS_Mod_1(Im,bufferImage,Mask,Mask_1,8,4);
imgComp = motionComp(bufferImage, motionVect, mbSize);
imgComp1 = motionComp_1(imgComp, motionVect1, 8);

if level==1
    imgComp=(imgComp+imgComp1)/2;
else
    imgComp=imgComp1;
end

imageSubtract=imsubtract_W(abs(Im),abs(imgComp));
bufferImageP=(imadd(abs(Im),abs(imgComp)));
bufferImageP= mirdwt(bufferImageP,yh1,h,1);
streamP=imageSubtract;


function decoder1(c,motionVect,identifier,flag)

global I1d P4d P7d I10d B2d B3d B5d B6d B8d B9d
```

```
blocksize=16;
i = 0;
h = daubcqf(6);
[ll_lev2,yh,L] = mrdwt(c,h,2);
c=ll_lev2;
if identifier==1
    I1d = Idecoder1(c,yh,h);
end

if identifier==4
    P4d = pDecoder1(c,yh,h,I1d,motionVect);
end

if identifier==7
    P7d = pDecoder1(c,yh,h,P4d,motionVect);
end

if identifier==2
    if flag==0
        B2d = bDecoder1(c,yh,h,I1d,motionVect);
    elseif flag==1
        B2d = bDecoder1(c,yh,h,P4d,motionVect);
    end
end

if identifier==3
    if flag==0
        B3d = bDecoder1(c,yh,h,I1d,motionVect);
    elseif flag==1
        B3d = bDecoder1(c,yh,h,P4d,motionVect);
    end
end

if identifier==5
    if flag==0
        B5d = bDecoder1(c,yh,h,P4d,motionVect);
    elseif flag==1
        B5d = bDecoder1(c,yh,h,P7d,motionVect);
    end
end

if identifier==6
    if flag==0
        B6d = bDecoder1(c,yh,h,P4d,motionVect);
    elseif flag==1
        B6d = bDecoder1(c,yh,h,P7d,motionVect);
    end
end

if identifier==10
    I10d = Idecoder1(c,yh,h);
    I1d=I10d;
```

```
end

if identifier==8
    if flag==0
        B8d = bDecoder1(c,yh,h,P7d,motionVect);
    elseif flag==1
        B8d = bDecoder1(c,yh,h,I10d,motionVect);
    end
end

if identifier==9
    if flag==0
        B9d = bDecoder1(c,yh,h,P7d,motionVect);
    elseif flag==1
        B9d = bDecoder1(c,yh,h,I10d,motionVect);
    end
end


% Computes motion compensated image using the given motion vectors
%
% Input
%   imgI : The reference image
%   motionVect : The motion vectors
%   mbSize : Size of the macroblock
%
% Ouput
%   imgComp : The motion compensated image
%
% Written by Aroh Barjatya

function imgComp = motionComp(imgI, motionVect, mbSize)

% imgI=P4;
% motionVect=motionVect1;
% mbSize=8;

[row col] = size(imgI);

% for i = mbSize:mbSize:row-mbSize-1
%     for j = mbSize:mbSize:col-mbSize-1
% we start off from the top left of the image
% we will walk in steps of mbSize
% for every marcoblock that we look at we will read the motion vector
% and put that macroblock from refernce image in the compensated image

mbCount = 1;
for i = 1:mbSize:row-mbSize+1
    for j = 1:mbSize:col-mbSize+1

        % dy is row(vertical) index
        % dx is col(horizontal) index
```

```
        % this means we are scanning in order

        dy = motionVect(1,mbCount);
        dx = motionVect(2,mbCount);

        refBlkVer = i + dy;
        refBlkHor = j + dx;

        imageComp(i:i+mbSize-1,j:j+mbSize-1) =
imgI(refBlkVer:refBlkVer+mbSize-1, refBlkHor:refBlkHor+mbSize-1);

        mbCount = mbCount + 1;
    end
end

imgComp = imageComp;

% Computes the Mean Absolute Difference (MAD) for the given two blocks
% Input
%       currentBlk : The block for which we are finding the MAD
%       refBlk : the block w.r.t. which the MAD is being computed
%       n : the side of the two square blocks
%
% Output
%       cost : The MAD for the two blocks
%
% Written by Aroh Barjatya


function cost = costFuncMAD(currentBlk,refBlk, n)

%currentBlk=double(currentBlk);
%refBlk=double(refBlk);
err = 0;
for i = 1:n
    for j = 1:n
        err = err + abs((currentBlk(i,j) - refBlk(i,j)));
    end
end
cost = err / (n*n);



function
[totaltime,avgMBSearch,avgMAD,avgMSE,PSNR]=HBMA(Target_Img,Anchor_Img,I
mg_Height,Img_Width,BlockSize,rangs,range,figureon)
%
%function
[totaltime,avgMBSearch,avgMAD,avgMSE,PSNR]=HBMA(Target_Img,Anchor_Img,I
mg_Height,Img_Width,BlockSize,rangs,range,figureon)
%
```

```
%This function calculate block motion vectors (with integer pel
accuracy), using hierarchical block matching algorithm.
%An example of main function calling this function is "MEMBA", which
can be entered on the command window.
%The function also use the function "EBMA" for motion estimation of
every macroblock
%
%   TargetName,AnchorName:
%       File Names of Target Frame and Anchor Frame
%   Img_Height,Img_Width:
%       Image Height and Width of a Frame
%  BlockSize:
%       The size of Macro Block in Frame is BlockSize(1) by
BlockSize(2)
%   rangs,range:
%       The Search Field in Frame A is from (rangs(1),rangs(2)) to
(range(1),range(2))
%  Target_Img,Anchor_Img,Predict_Img:
%       Image Matrix for Target Frame, Anchor Frame, Predicted Frame
%   ox,oy,pxx,pyy:
%       The location of Motion vector is (ox,oy), (pxx,pyy) for the
direction
%   PSNR:
%       The peak signal and noise ratio between original image and
predicted image
%   L:
%       The search level
%   Author: Xiaofeng Xu, Polytechnic University  4/21/2002
%   totaltime:
%       The total time of ME algorithm execution between original and
predicted images (platform depended)
%   avgMBSearch:
%        The average number of Macro Block matching stages between
original and predicted images
%   avgMAD:
%       The average MAD between original and predicted images
%   avgMSE:
%       The average MSE between original and predicted images
%   Author: Evgeny Kaminsky, Ben Gurion University 12/18/2002
L=3;
%Number of MB searches;
c_MB_search=0;
%Read images from files
%fid = fopen(Target_Img,'r');
%Target_Img= fread(fid,[Img_Height,Img_Width]);
%fclose(fid);
Target_Img=double(Target_Img);

%fid = fopen(Anchor_Img,'r');
%Anchor_Img= fread(fid,[Img_Height,Img_Width]);
%fclose(fid);
Anchor_Img=double(Anchor_Img);
```

```
if (figureon)
    %Display the results
    figure;
    imshow(uint8(Target_Img));
    title('Target Image')
end
t0 = clock;
m=1;
Factor=2.^(L-1);
%Downsample Image with different resolution
Up_Target_Img=zeros(Img_Height*2,Img_Width*2);
Up_Target_Img(1:2:Img_Height*2,1:2:Img_Width*2)=Target_Img;
Up_Target_Img(1:2:Img_Height*2-1,2:2:Img_Width*2-
1)=(Target_Img(:,1:Img_Width-1)+Target_Img(:,2:Img_Width))/2;
Up_Target_Img(2:2:Img_Height*2-1,1:2:Img_Width*2-
1)=(Target_Img(1:Img_Height-1,:)+Target_Img(2:Img_Height,:))/2;
Up_Target_Img(2:2:Img_Height*2-1,2:2:Img_Width*2-
1)=(Target_Img(1:Img_Height-1,1:Img_Width-1)+Target_Img(1:Img_Height-
1,2:Img_Width)+Target_Img(2:Img_Height,1:Img_Width-
1)+Target_Img(2:Img_Height,2:Img_Width))/4;

TargetDown=zeros(3,Img_Height,Img_Width);
%AnchorDown=TargetDown;
TargetDown1=Target_Img;
AnchorDown1=Anchor_Img;

AnchorDown2(1:Img_Height/2,1:Img_Width/2)=Anchor_Img(1:2:Img_Height,1:2
:Img_Width);
AnchorDown3(1:Img_Height/4,1:Img_Width/4)=AnchorDown2(1:2:Img_Height/2,
1:2:Img_Width/2);

TargetDown2(1:Img_Height/2,1:Img_Width/2)=Target_Img(1:2:Img_Height,1:2
:Img_Width);
TargetDown3(1:Img_Height/4,1:Img_Width/4)=TargetDown2(2:2:Img_Height/2,
1:2:Img_Width/2);

Predict_Img=Target_Img;

rangs(1)=rangs(1)/Factor;
range(1)=range(1)/Factor;

rangs(2)=rangs(2)/Factor;
range(2)=range(2)/Factor;

Img_Height=Img_Height/Factor;
Img_Width=Img_Width/Factor;


%Search for all the blocks in Anchor Images of 1st level
for i=1:BlockSize(1):Img_Height-BlockSize(1)+1
   RangeStart(1)=i+rangs(1);
```

```
   RangeEnd(1)=i+BlockSize(1)-1+range(1);
   if RangeStart(1)<1
      RangeStart(1)=1;
   end
   if RangeEnd(1)>Img_Height
      RangeEnd(1)=Img_Height;
   end
   for j=1:BlockSize(2):Img_Width-BlockSize(2)+1
      RangeStart(2)=j+rangs(2);
      RangeEnd(2)=j+BlockSize(2)-1+range(2);
       if RangeStart(2)<1
       RangeStart(2)=1;
       end
    if RangeEnd(2)>Img_Width
        RangeEnd(2)=Img_Width;
      end
      tmpt(:,:)=TargetDown3(:,:);
      tmpa(:,:)=AnchorDown3(:,:);
      [px(m),
py(m),MB_search]=EBMA(tmpt,tmpa,BlockSize,[i,j],RangeStart,RangeEnd);
      c_MB_search=MB_search+c_MB_search;
      ox(m)=j;
      oy(m)=i;
      m=m+1;
   end
end
if (figureon)
%Disfplay the results
    figure;
    imshow(uint8(TargetDown3));
    title('TargetDown3')
    figure;
    imshow(uint8(AnchorDown3));
    title('AnchorDown3')

    hold on
    quiver(ox,oy,px,py);

    hold off
    axis image
end
%Search for all the blocks in Anchor Images of all levels
for ii=L-1:-1:1
   %Update all parameters for the currenet level.
   px=px*2;
   py=py*2;
   Img_Height=Img_Height*2;
   line_width=floor(Img_Width/BlockSize(2));
   Img_Width=Img_Width*2;
   ttt=size(py);

    m=1;
    %Search for all the blocks in Anchor Images in the iith level
```

```
    for i=1:BlockSize(1):Img_Height-BlockSize(1)+1

      baseline=double(uint32(i/2/BlockSize(1)))*double(line_width);
      for j=1:BlockSize(2):Img_Width-BlockSize(2)+1
         %Caculate the search range in Target Images.
         mindx=floor(baseline+double(uint32(j/2/BlockSize(2)))+1);
         if mindx>ttt(2)
            mindx=ttt(2);
         end

         RangeStart(1)=i+py(mindx)+rangs(1);
        RangeEnd(1)=i+py(mindx)+BlockSize(1)-1+range(1);
        if RangeStart(1)<1
        RangeStart(1)=1;
        end
        if RangeEnd(1)>Img_Height
        RangeEnd(1)=Img_Height;
        end

         RangeStart(2)=j+px(mindx)+rangs(2);
          RangeEnd(2)=j+px(mindx)+BlockSize(2)-1+range(2);
         if RangeStart(2)<1
           RangeStart(2)=1;
           end
        if RangeEnd(2)>Img_Width
            RangeEnd(2)=Img_Width;
         end

         if ii==2
             tmpt=TargetDown2(:,:);
              tmpa=AnchorDown2(:,:);

         end

         if ii==1
             tmpt=TargetDown1(:,:);
              tmpa=AnchorDown1(:,:);

          end

                [pxx(m), pyy(m),MB_search,
Predict_Img(i:i+BlockSize(1)-1,j:j+BlockSize(1)-
1)]=EBMA(tmpt,tmpa,BlockSize,[i,j],RangeStart,RangeEnd);
                c_MB_search=MB_search+c_MB_search;

                  %Refine final result by half-pel accuracy search
         if(ii==1)
            RangeStart(1)=(i+pyy(m))*2-1-2;
                RangeEnd(1)=(i+pyy(m))*2-1+BlockSize(1)*2-1+2;
            if RangeStart(1)<1
            RangeStart(1)=1;
            end
```

```
            if RangeEnd(1)>Img_Height*2
            RangeEnd(1)=Img_Height*2;
            end

            RangeStart(2)=(j+pxx(m))*2-1-2;
              RangeEnd(2)=(j+pxx(m))*2-1+BlockSize(2)*2-1+2;
          if RangeStart(2)<1
               RangeStart(2)=1;
            end
            if RangeEnd(2)>Img_Width*2
               RangeEnd(2)=Img_Width*2;
           end
              tmpa=AnchorDown1(:,:);
                [pxx(m), pyy(m),MB_search,Predict_Img(i:i+BlockSize(1)-
1,j:j+BlockSize(1)-
1)]=EBMA(Up_Target_Img,tmpa,BlockSize,[i,j],RangeStart,RangeEnd,2);
                c_MB_search=MB_search+c_MB_search;
      end
            ox(m)=j;
          oy(m)=i;
        m=m+1;
      end
    end
   px=pxx;
   py=pyy;

end
totaltime=etime(clock,t0);
imgsize = Img_Height*Img_Width;
%Caculate error image
Error_Img=Anchor_Img-Predict_Img;
%Calculate totalerror
totalerror=sum(sum(abs(Error_Img)));
%Calculate average MAD
avgMAD=totalerror/imgsize;
%Calculate average MSE
avgMSE=mean(mean((Error_Img.^2)));
%Caculate PSNR
PSNR=10*log10(255*255/avgMSE);
%Claculate average number of searching stages for each Macro Block
MB_total=imgsize/(BlockSize(1)*BlockSize(2));
avgMBSearch = c_MB_search/MB_total;


function [Out] = Idecoder1(mat,yh,h);

sphit_main_encode
sphit_main_decode

%-------------------------------------------------------
% Reference : A New, Fast, Efficient Image Codec using Set Partitioning
of Hierarchical Trees
```

```
%:- Amir. Said, W. A Pearlman
%-------------------------------------------------------------------------
------
function sphit_main_encode
load indices
format short
mat=trans_1(x,'bior3.7');
mat=fix(mat);
seqt(xm)=mat;
T=(2^fix(log2(max(max(abs(mat))))))/2;
global maxy
maxy=T;
LIS=[];
%
=========================================================================
======
% Initial lists
for jh=65:256
    LIS=[LIS jh 0];%[2 0 3 0 4 0];    % List of Insignificant Sets ,A-0
& B-1, Co-ordinates
end
LIP=[1:256];%[1 2 3 4];          % List of Insignificant Pixels, Co-
ordinates
LSP=[];               % List of Significant Pixels, Co-ordinates
output=[];

% Initializaton complete, Starting processing
bit_number=1;

for xx=1:8

    sendlsp=LSP;
%-------------------------------------------------------------------------
for ii=1:length(LIP) % i.e for each entry in the LIP, do
    if abs(seqt(LIP(ii)))>=T
        output=[output '1'];
        LSP=[LSP LIP(ii)]; % Moving ii to LSP, Removing ii from LSP
done later
        % Output sign
        if seqt(LIP(ii))>=0
            output=[output '0'];
        else
            output=[output '1'];
        end
    else
        output=[output '0'];
    end
end
%-------------------------------------------------------------------------
% Now remove the common elements i.e perform LIP-LSP
% D(i,j) means all descendants of (i,j), Function Descendants does this
% O(i,j) means offsprings of (i,j), Function offspring does this
% L(i,j) = D(i,j) - O(i,j), Done by mark_proper
```

```
LIP=mark_proper(LIP,LSP);
track=[];
ij=1;
while ij<=length(LIS)
%for ij=1:2:length(LIS) % For each entry in LIS

    if LIS(ij+1)==0 % A type

        out=0;out1=0;% LIS(ij)  % problem
        %-----------------------------------------------------------------
        if ~isempty(find(abs(seqt(descendants_1(LIS(ij))))>=T))
% Check for offsprings of ii
            output=[output '1'];
            out=1;
        else
            output=[output '0'];
        end

        if out==1
            % Star 1
            var1=offspring_1(LIS(ij));
            for kl=1:4
                if abs(seqt(var1(kl)))>=T
                    output=[output '1'];
                    out1=1;
                else
                    output=[output '0'];
                end

                if out1==1
                    LSP=[LSP var1(kl)];

                    % sign
                    if seqt(var1(kl))>=0
                        output=[output '0'];
                    else
                        output=[output '1'];
                    end

                else
                    LIP=[LIP var1(kl)];
                end
                out1=0;
            end

            % Star 2

lij=mark_proper(descendants_1(LIS(ij)),offspring_1(LIS(ij)));

            if ~isempty(lij)
                % Move ij to the end of LIS as an entry of type B
```

```
                LIS=[LIS LIS(ij) 1];
                track=[track ij];
            else
                track=[track ij];
            end
        end
    end

    if LIS(ij+1)==1 % B type
        out=0;
        lij=mark_proper(descendants_1(LIS(ij)),offspring_1(LIS(ij)));
        if ~isempty(find(abs(seqt(lij))>=T))
            output=[output '1'];
            out=1;
        else
            output=[output '0'];
        end

        if out==1
            var1=offspring_1(LIS(ij));
            for mn=1:4
                LIS=[LIS var1(mn) 0];
            end
            track=[track ij];
        end
        out=0;
    end
    ij=ij+2;
end
% ------------------------------------------------
% Remove repeating elements
if ~isempty(LIS)

    for z=1:length(track)
        LIS(track(z):track(z)+1)=9999;
    end
    % ------------------------------------------------
    LIS=LIS(find(LIS~=9999));

end

if ~isempty(sendlsp)
    output=refinement(output,sendlsp,seqt,bit_number);
end
bit_number=bit_number+1;
[  T  length(LSP) length(find(abs(seqt)>=T))]
T=T/2;
end

disp(' OVER ')
save filename output
% output
```

```
%LSP
%LIP
%LIS

%-------------------------------------------------------------------------
% Reference : A New, Fast, Efficient Image Codec using Set Partitioning
of Hierarchical Trees
%:- Amir. Said, W. A Pearlman
%-------------------------------------------------------------------------
function sphit_main_decode
load filename output
format long
load indices

orig=x;
T=1024;  % For barbara 2048 % else 1024
mmx=T;
xm=mapping_256;
xm=xm(:);
%=========================================
mat=trans_1(x,'bior3.7');
mat=fix(mat);
dect(xm)=mat;  % Original
%=========================================
seqt(256*256)=0;
global iii
iii=0;
LIS=[];
%
=========================================================================
% Initial lists
for jh=65:256
    LIS=[LIS jh 0];%[2 0 3 0 4 0]; % List of Insignificant Sets ,A-0 &
B-1, Co-ordinates
end
LIP=[1:256];%[1 2 3 4];        % List of Insignificant Pixels, Co-
ordinates
LSP=[];                        % List of Significant Pixels, Co-
ordinates

% Initializaton complete, Starting processing

for xx=1:8

    getlsp=LSP;
%-------------------------------------------------------------------------
for ii=1:length(LIP) % i.e for each entry in the LIP, do
    bit=input_bit;
    if bit=='1'
        LSP=[LSP LIP(ii)];
        bit=input_bit;
        if bit=='1'
            seqt(LIP(ii))=-mean([T 2*T]);
```

```
        else
            seqt(LIP(ii))=mean([T 2*T]);
        end
    end
end
%-------------------------------------------------------------------------
% Now remove the common elements i.e perform LIP-LSP
% D(i,j) means all descendants of (i,j), Function Descendants does this
% O(i,j) means offsprings of (i,j), Function offspring does this
% L(i,j) = D(i,j) - O(i,j), Done by mark_proper

LIP=mark_proper(LIP,LSP);

track=[];
ij=1;
while ij<=length(LIS)
%for ij=1:2:length(LIS) % For each entry in LIS

    if LIS(ij+1)==0 % A type

        out=0;out1=0;% LIS(ij)  % problem
        %-----------------------------------------------------------------
        bit=input_bit;
        if bit=='1' % Check for offsprings of ii
            out=1;
        end

        if out==1
            % Star 1
            var1=offspring_1(LIS(ij));
            for kl=1:4
                bit=input_bit;
                if bit=='1'
                    out1=1;
                end

                if out1==1
                    LSP=[LSP var1(kl)];

                    % sign
                    bit=input_bit;
                    if bit=='0'
                        seqt(var1(kl))=mean([T 2*T]);
                    else
                        seqt(var1(kl))=-mean([T 2*T]);
                    end

                else
                    LIP=[LIP var1(kl)];
                end
                out1=0;
            end
```

```
            % Star 2

lij=mark_proper(descendants_1(LIS(ij)),offspring_1(LIS(ij)));

            if ~isempty(lij)
                % Move ij to the end of LIS as an entry of type B
                LIS=[LIS LIS(ij) 1];
                track=[track ij];
            else
                track=[track ij];
            end
        end
    end

    if LIS(ij+1)==1 % B type
        out=0;
        lij=mark_proper(descendants_1(LIS(ij)),offspring_1(LIS(ij)));
        bit=input_bit;
        if bit=='1'
            out=1;
        end

        if out==1
            var1=offspring_1(LIS(ij));
            for mn=1:4
                LIS=[LIS var1(mn) 0];
            end
            track=[track ij];
        end
        out=0;
    end
    ij=ij+2;
end

if ~isempty(LIS)
    % -----------------------------------------------------
    % Remove repeating elements
    for z=1:length(track)
        LIS(track(z):track(z)+1)=9999;
    end
    % -----------------------------------------------------
    LIS=LIS(find(LIS~=9999));
end

if ~isempty(getlsp)
    seqt=irefinement(seqt,T,getlsp);
end

% rec=round(reshape(seqt(xm(:)),128,128));
rec=round(reshape(seqt(xm(:)),256,256));
%xr=idwt2d(rec,fopt,4);
```

```
xr=itrans_1(rec,'bior3.7');
%axis tight
set(gca,'nextplot','replacechildren');
figure%,imshow(mat2gray(fix(xr)))
% Record the movie
    imshow(mat2gray(fix(xr)));
    F = getframe;

% Play the movie ten times
movie(F)
[snr , msr]=PSNR(xr,orig);
% ****************************************************
format short g
[T iii snr msr max(abs(abs(seqt)-abs(dect)))]
format long
% ****************************************************
T=T/2;
end
%round(reshape(seqt(x(:)),128,128))


function [mvX mvY] = mvFrame(tFrame,fFrame,mbSize,limitSad,sadLimit)
% Configuration
% Perform sequential search, log search, or hierarchical search
sType = 0; % 0 = sequential search
% 1 = log search
% 2 = hierarchical search
% Default size to search over
stepSize = 64;
[vPixel hPixel] = size(fFrame);
for hPos = 16:mbSize:hPixel
    for vPos = 16:mbSize:vPixel
        [mvX(vPos/mbSize,hPos/mbSize) mvY(vPos/mbSize,hPos/mbSize)
minVal] = ...
            mvMacroblock(tFrame(vPos-15:vPos,hPos-
15:hPos),fFrame,mbSize, ...
            hPos,vPos,stepSize,sType);
        if limitSad && minVal > sadLimit
            % The motion vector search could not find a "good enough"
            % estimate. Ignore the results.
            mvX(vPos/mbSize,hPos/mbSize) = inf;
            mvY(vPos/mbSize,hPos/mbSize) = inf;
        end
    end
end


function [xVec yVec minVal] = mvMacroblock(mb, fFrame, mbSize, hPos,
vPos, stepSize, sType)
[vPixel hPixel] = size(fFrame);
global l1Frame;
global l2Frame;
if sType == 0
```

```
    % Sequential search
    x = hPos - mbSize + 1;
    y = vPos - mbSize + 1;
    minVal = inf;
    for distance = 0:stepSize-1
        if distance == 0
            mvXNdx = 0; mvYNdx = 0;
        else
            mvXNdx = [ones(1,distance*2+1)*distance
ones(1,distance*2+1)*-distance ...
                (-distance+1:distance-1) (-distance+1:distance-1)];
            mvYNdx = [-distance:distance -distance:distance
ones(1,distance*2-1)*distance ...
                ones(1,distance*2-1)*-distance];
        end
        x1 = mvXNdx + x;
        x2 = mvXNdx + x + mbSize - 1;
        y1 = mvYNdx + y;
        y2 = mvYNdx + y + mbSize - 1;
        delNdx = find(x1 <= 0 | x2 > hPixel | y1 <=0 | y2 > vPixel);
        x1(delNdx) = []; x2(delNdx)=[]; y1(delNdx)=[]; y2(delNdx)=[];
        for sadNdx = 1:length(x1)
            val = sum(sum(abs(mb-fFrame(y1(sadNdx):y2(sadNdx),...
                x1(sadNdx):x2(sadNdx)))));
            if val < minVal
                minVal = val;
                xVec = x1(sadNdx)-x; yVec = y1(sadNdx)-y;
            end
        end
    end
elseif sType == 1
    % Logarithmic search
    % The search vector order below makes sure we take the shortest
distance
    % in the case of a tie
    stepSize = stepSize/2;
    x = hPos; y = vPos;
    sVect = [0 0 0 1 -1 1 -1 1 -1; 0 1 -1 0 0 1 1 -1 -1];
    while stepSize >= 1
        sad = ones(1,9)*inf;
        for sVectLoc = 1:9
            x2 = sVect(1,sVectLoc) * stepSize + x;
            x1 = sVect(1,sVectLoc) * stepSize + x - mbSize + 1;
            y2 = sVect(2,sVectLoc) * stepSize + y;
            y1 = sVect(2,sVectLoc) * stepSize + y - mbSize + 1;
            if x1 <= 0 || x2 > hPixel || y1 <= 0 || y2 > vPixel
                continue;
            else
                sad(sVectLoc) = sum(sum(abs(mb-fFrame(y1:y2,x1:x2))));
            end
        end
        [dummy ndx] = min(sad);
        ndx = ndx(1);
        x = x + sVect(1,ndx) * stepSize;
```

```
        y = y + sVect(2,ndx) * stepSize;
        stepSize = stepSize/2;
    end
    xVec = x - hPos;
    yVec = y - vPos;
elseif sType == 2
    % Hierarchical search
    l1Mb = mb(1:2:end,1:2:end);
    l2Mb = l1Mb(1:2:end,1:2:end);
    l2hPos = hPos/4; l2vPos = vPos/4;
    [mvX2 mvY2] = mvMacroblock(l2Mb, l2Frame, mbSize/4, l2hPos, l2vPos,
stepSize/4, 0);
    l1hPos = l2hPos*2 + mvX2*2; l1vPos = l2vPos*2 + mvY2*2;
    [mvX1 mvY1] = mvMacroblock(l1Mb, l1Frame, mbSize/2, l1hPos, l1vPos,
2, 0);
    l0hPos = l1hPos*2 + mvX1*2; l0vPos = l1vPos*2 + mvY1*2;
    [mvX mvY] = mvMacroblock(mb, fFrame, mbSize, l0hPos, l0vPos, 2, 0);
    xVec = mvX2*4 + mvX1*2 + mvX;
    yVec = mvY2*4 + mvY1*2 + mvY;
else
    error('Invalid search type');
end
x = hPos - mbSize + 1 + xVec;
y = vPos - mbSize + 1 + yVec;
minVal = sum(sum(abs(mb-fFrame(y:y+mbSize-1,x:x+mbSize-1))));



function [JT,DT,RT]= lagrangian_cost(I,lambda)

% Transfer I frame into RDWT
h = daubcqf(6);
[ll_lev2,yh,L] = mrdwt(I,h,1);
N = 256;
lh = yh(:,1:N);
hl = yh(:,N+1:2*N);
hh = yh(:,2*N+1:3*N);
img = ll_lev2;
dim = size(img,1);
step =16;                       %% Step Size
%   *(1) = 16x16 blocks
%   *(2) = 8x8 blocks
%   *(3) = 4x4 blocks
M = [16, 8, 4]                 %% Size of block
B_cnt = dim./M          %% # of blocks per image
B_opt = size(M,2)       %% # of block sizes available
img_recon = zeros(dim,dim);
img_final = zeros(dim,dim);
%lambda = 0
lambda_sz = size(lambda, 2);
J = zeros(B_cnt(1),B_cnt(1));     %% Block Cost Function
D = zeros(B_cnt(1),B_cnt(1));     %% Block Distortion
R = zeros(B_cnt(1),B_cnt(1));     %% Block Rate
```

```
% QUADTREE CODE-
%   | 0 | 0 0 0 0 | - use 16x16 block (0 bits per 16x16 pixels)
%   | 1 | 0 0 0 0 | - use 4 8x8 blocks (1 bit per 16x16 pixels)
%   | 1 | 1 0 0 0 | - use 3 8x8 blocks & 4 4x4 blocks (5 bits per 16x16
pixels)
%   | 1 | 1 0 1 0 | - use 3 8x8 blocks & 4 4x4 blocks (5 bits per 16x16
pixels)
QTcode = zeros(B_cnt(1),5,B_cnt(1),lambda_sz);
% per pixel QTrate
QTrate = [0,1,5]/(M(1)*M(1));
B16 = zeros(M(1),M(1));          % 16x16 block
B8  = zeros(M(2), M(2),4);       % 8x8 block
B4  = zeros(M(3), M(3),4);       % 4x4 block

for w=1:lambda_sz

   % Loop through each 16x16 block and make the decision whether or
   %  not to break down the block into 8x8 and/or 4x4 blocks

   for i=1:B_cnt(1)
      for j=1:B_cnt(1)
         row = M(1)*(i-1);
         col = M(1)*(j-1);

         % Create  16x16 block
         B16(:,:) = img(row+1:row+M(1),col+1:col+M(1));

         %Subdivide the 16x16 block into 4 8x8 blocks
         B8(:,:,1) = img(row+1:row+M(2), col+1:col+M(2));
         B8(:,:,3) = img(row+1:row+M(2), col+1+M(2):col+2*M(2));
         B8(:,:,2) = img(row+1+M(2):row+2*M(2), col+1:col+M(2));
         B8(:,:,4) = img(row+1+M(2):row+2*M(2), col+1+M(2):col+2*M(2));

     % For 16x16 block, find the distortion, bitrate and cost function
         [D16, R16, B_Cost16 ] = blk_calc(B16, step);
         J16 = D16 + lambda(w).*R16;

   % For each 8x8 block, find the distortion, bitrate and cost function
         for p=1:4
            [D8(p), R8(p), B_Cost8(:,:,p)] = blk_calc(B8(:,:,p), step);
            J8(p)  = D8(p) + lambda(w).*R8(p);
         end

     % Find the average distortion,bitrate, & cost for the 4 8x8 blocks
         % Note that for the cost, you must add the additional bits for
         %   the quadtree code
         J8_QT = sum(J8)/4 + lambda(w)*QTrate(2);

         % Compare costs; if the cost of 4 8x8 blocks is less than the
         %  the cost of a single 16x16 block, break up the block
         if J16 > J8_QT
```

```
            % Update the quadtree code
            QTcode(i,1,j,w) = 1;

        % Now, compare each 8x8 block with its corresponding 4x4 blocks
            for q=1:4

               %Subdivide the 8x8 block into 4 4x4 blocks
               B4(:,:,1) = B8(1:4,1:4,q);
               B4(:,:,2) = B8(1:4,5:8,q);
               B4(:,:,3) = B8(5:8,1:4,q);
               B4(:,:,4) = B8(5:8,5:8,q);

   % For each 4x4 block, find the distortion, bitrate and cost function
               for p=1:4
              [D4(p,q), R4(p,q), B_Cost4(:,:,p)] = blk_calc(B4(:,:,p),
step);
                  J4(p,q) = D4(p,q) + lambda(w).*R4(p,q);
               end

               J4_QT =  sum(J4(:,q))/4 + lambda(w)*QTrate(3);

          % Compare costs; if the cost of 4 4x4 blocks is less than the
               %  the cost of a single 8x8 block, break up the block
               if J8(q) > J4_QT

                  % Update the quadtree code
                  QTcode(i,1+q,j,w) = 1;

                  % Create a "new" 8x8 block made up of 4x4 blocks
                  B8_new(1:4,1:4,q) = B_Cost4(:,:,1);
                  B8_new(1:4,5:8,q) = B_Cost4(:,:,2);
                  B8_new(5:8,1:4,q) = B_Cost4(:,:,3);
                  B8_new(5:8,5:8,q) = B_Cost4(:,:,4);
                  J8_new(q) = J4_QT;
                  D8_new(q) = sum(sum(D4(:,q)))/4;
                  R8_new(q) = sum(sum(R4(:,q)))/4 + QTrate(3);

               % If it costs less to use this 8x8 block, don't split it
               else
                  B8_new(:,:,q) = B_Cost8(:,:,q);
                  J8_new(q) = J8(q);
                  D8_new(q) = D8(q);
                  R8_new(q) = R8(q) + QTrate(2);
               end
            end % for q

            % Create a "new" 16x16 block made up of 8x8 blocks
            B16_new(1:8,1:8) = B8_new(:,:,1);
            B16_new(1:8,9:16) = B8_new(:,:,2);
            B16_new(9:16,1:8) = B8_new(:,:,3);
            B16_new(9:16,9:16) = B8_new(:,:,4);
```

```
            % Store the cost, distortion and rate for the new image
            J(i,j) = sum(J8_new)/4;
            D(i,j) = sum(D8_new)/4;
            R(i,j) = sum(R8_new)/4;

            % Add the created 16x16 block to the reconstructed image
            img_recon(row+1:row+M(1),col+1:col+M(1)) = B16_new;

         % If it costs less to use the 16x16 block, don't split it
         else

            % Add the 16x16 block to the reconstructed image
            img_recon(row+1:row+M(1),col+1:col+M(1)) = B_Cost16;

            J(i,j) = J16;
            D(i,j) = D16;
            R(i,j) = R16;
         end
         % Update Lamba
          lambda=max(D(i,j))-Min(D(i,j))/max(R(i,j))-Min(R(i,j));
      end  % for j
   end  % for i

   % find the total cost function, rate and distortion for the entire
image
   %   (4x4, 8x8, 16x16, best)
   JT(w) = sum(sum(J(:,:)))/(B_cnt(1)*B_cnt(1))
   DT(w) = sum(sum(D(:,:)))/(B_cnt(1)*B_cnt(1))
   RT(w) = sum(sum(R(:,:)))/(B_cnt(1)*B_cnt(1))
end % f


function [Dist, Rate,B_Cost] = blk_calc(B, step)

  % PER BIT VALUES FOR DISTORTION & RATE
   M = size(B, 2);
   h = daubcqf(6);
   [ll_lev2,yh,L] = mrdwt(I1,h,1);
   B_LL = ll_lev2;
   B_q = quant(B_LL, step);
   B_Cost = mirdwt(B_q,yh,h,1);
   % Find the average distortion
   Dist = sum(sum((B_q - B_LL).^2))/(M*M);
   % Find the B_q bitrate
   range = max(max(B_q)) - min(min(B_q));
   pmf = zeros(range, 1);
   temp_pmf = hist(B_q, range);
   for n=1:size(temp_pmf,2)
      pmf = temp_pmf(:,n) + pmf;
   end
   pmfsize = size(pmf);
   pmf = pmf/sum(pmf);
```

```
% bits/pixel*M*M = bits/B_q
Rate = sum( -pmf.*log2(pmf + (pmf ==0)) );
```