New Hybrid Coding for Video Sequences

WONG Kai Yin

A thesis
submitted in partial fulfillment of the requirements
for the Degree of Master of Philosophy
in the Department of Electronic and Information Engineering

The Hong Kong Polytechnic University

October 2006

Certificate of Originality

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made n the text.

_____(Signed)

_____WONG KAI-YIN_____(Name of student)

Abstract

During the encoding process of traditional video coding standards, the motion estimation consumes most of the computational effort. For the wavelet video coding, this problem becomes more serious since motion estimation in the wavelet domain involves floating-point computation. Hence, it is necessary to develop some fast motion estimation algorithms in wavelet domain in order to reduce the computational burden of the wavelet video encoder during encoding procedure.

One of the traditional motion estimation algorithms in wavelet domain is to make use of the correlations among the corresponding subbands in the wavelet pyramid to enhance the speed of motion estimation. This algorithm is entitled as Multi-resolution Motion Estimation (MRME) algorithm which is based on the fact that an object in a subband of the lowest resolution level actually specifies the same object in the subbands of the higher resolution levels. Thus, the computational complexity of motion estimation in the wavelet domain can be reduced significantly by exploiting the relationships between the subbands of different resolution levels. On the other hand, the pixels with similar matching error tend to group in a cluster in both spatial and wavelet domains. Besides, a cluster which appears in a certain position of a subband at the lowest resolution level and there also exists a cluster in the similar position of the corresponding subbands at the remaining higher resolution levels. Thus, the Clustered Pixel Matching Error for Partial Distortion Search (CPME-PDS) algorithm can be applied in the MRME scheme to further improve the speed of motion estimation in the wavelet domain. The CPME-PDS involves the sorting procedure in order to obtain the coefficients with large matching errors and the sorting is required to perform in every

subband at each decomposition level. As we found that the clustering property is in a hierarchical nature in the wavelet pyramid, the sorting order in the subband of the highest-resolution level can be re-used to predict the sorting order of the subbands in the lower-resolution levels. Since the sorting operations are only carried out in the three subbands at the high-resolution level only, so the computational effort for motion estimation can be further reduced. From the experimental results, the proposed algorithm (Backward CPME-PDS) can achieve speed-up factors from 2 to 5 and from 1.1 to 1.2 as compared to the Full Search Algorithm (FSA) and Partial Distortion Search (PDS) algorithm respectively.

Recently, the three dimensional discrete wavelet transform (3D-DWT) video coder becomes more popular since it can attain both spatial and temporal scalabilities. It involves motion estimation in the wavelet decomposition, so the computational burden for motion estimation is still a major concern. Due to the fact that there exists high spatial and temporal correlations between the motion vectors in the neighbouring blocks in a frame and between the motion vector fields of the low frequency frames at successive temporal levels, the computational complexity of motion estimation can be reduced considerably. By making use of these correlations, an accurate motion estimation predictor can be obtained and a refinement process is performed within the reduced search area based on the accurate motion predictor. From the experimental results, the proposed algorithm can achieve a speed-up factor of 3 to 5 as compared with the FSA using the Haar and Bi-orthogonal 5/3 kernels during temporal decomposition. Besides, quality of the reconstructed video sequence using the proposed algorithm is comparable to that of the FSA.

The Embedded Zerotree Wavelet (EZW) coding algorithm is often used to encode the wavelet coefficients. We proposed a modified EZW algorithm to

improve the coding gain of the wavelet video coder by discarding some less important wavelet coefficients using a new criterion. Experimental results show that the proposed EZW algorithm can improve 0.2 to 0.5 bit per pixel (bpp) as compared with the original EZW algorithm for the same PSNR value in lossy coding and its reconstructed quality is comparable to that of the conventional EZW algorithm. Furthermore, the proposed algorithm can apply in the Set Partition Embedded Block Coding (SPECK) algorithm to further enhance the compression efficiency from 0.1 to 0.5 bit per pixel (bpp) as compared with the minimum subband approach applied in the SPECK algorithm for the same PSNR value in lossy coding and its visual quality of the reconstructed image using the proposed algorithm can be preserved.

Acknowledgments

I would like to take this opportunity to express my sincere gratitude to my Supervisor, Professor W.C. Siu, for his continuous encouragement, guidance and care during the period that I worked on this thesis. He gave me informative suggestions and valuable advice contributing to every success of my research. More importantly, I am deeply impressed with his hard working style and his willingness to devote to the advancement of science and research. This gives me a clear image of a great researcher should be and have inspired me to work hard on the thesis. It is beyond doubt that this will continuously influence my future research and career.

I would again like to express my sincere thank to Dr. Kenneth Lam, Dr. Y. L. Chan, Dr. Bonnie Law, Dr. K. T. Fung, Dr. K. C. Hui, Mr. W. L. Hui, Mr. H. K. Cheung, Mr. H. K. Cheung, Miss K. M. Au, Mr. W. P. Choi, Mr. K. C. Lui, Mr. K. H. Chung, Mr. Y. H. Kam, Mr. W. H. Wong, Mr. J. Xie, Mr. H. Pong, Mr. K. O. Cheng and Mr. C. W. Hui. The sharing of ideas and experience with them has greatly contributed to make every success of my work. It has been a wonderful time to me in these years to work with them.

Meanwhile, I am glad to express my gratitude to the Department of Electronic and Information Engineering and the Center of Multimedia Signal Processing for providing me a comfortable working environment, and the Hong Kong Polytechnic University for their generous financial support to carry out my research work.

Above all, I am deeply grateful to my family for their constant love, encouragement and support. Without their understanding and patience, it is impossible for me to complete this research study.

Statements of Originality

The following points are claimed to be original in this work.

1. During performing motion estimation in the spatial domain, the pixels with similar absolute matching errors tend to cluster together. This clustering property also exists in the wavelet domain. Furthermore, the clustering property in the wavelet domain is highly correlated among the corresponding subbands in the wavelet pyramid. According to this observation, the backward Clustered Pixel Matching Error for Partial Distortion Search (CPME-PDS) is proposed to enhance the speed of motion estimation in the wavelet domain. This is achieved by using the sorting order for performing the partial Sum of Absolute Difference (SAD) calculation in the subbands at the highest resolution level to predict the corresponding subbands at the lower resolution levels. Therefore, the number of operations used for sorting at each low-resolution subband can be reduced. Experimental results show that the proposed algorithm can outperform the Full Search Algorithm (FSA) by the speed-up factors of 1.97 and 5.58 in terms of actual implementation time and average number of operations respectively, without making use of other fast algorithms. More details can be found in section 3.2.
2. The Successive Elimination Algorithm (SEA) is employed in the backward CPME-PDS in order to further reduce the number of operations for motion estimation in the wavelet domain. Experimental results show that the proposed algorithm can achieve the speed-up factors of 2.16 and 10.30 in terms of total execution time and average number of operations respectively as compared with

the backward CPME-PDS. However, this improvement only exists in slow video sequence. For other video sequences, although the average number of operations can be reduced, the execution time is enlarged due to the increasing number of operations for comparison of floating point numbers. More details can be found in section 3.4.

3. Usually, the motion estimation is absorbed into the lifting steps of the one dimensional Discrete Wavelet Transform (1D-DWT) along the temporal direction in the conventional three dimensional Discrete Wavelet Transform (3D-DWT) video coder in order to improve the visual quality of the low frequency frames and the compression efficiency. In the traditional 3D-DWT video coder, the motion vectors are obtained by performing motion estimation in each temporal level independently. However, there exists a large correlation between the low frequency frames in the successive temporal levels. By exploiting of this redundancy, the motion vectors in the previous temporal level are used as initial search position and the motion vectors in the current temporal level are refined in the reduced search range so that the speed of motion estimation can be enhanced due to the reduction of number of operations in the condensed search window. Besides, the median motion vector approach is also employed in the first temporal level, i.e. the spatial domain, to perform motion estimation, so the execution time used for motion estimation can be further decreased. Experimental results show that the proposed scheme can achieve the speed-up factors of 5.53 and 5.59 using the Haar and Bi-orthogonal 5/3 kernels respectively as compared with FSA. More details can be found in section 4.2.

4. The Embedded Zerotree Wavelet (EZW) is usually used to encode the wavelet coefficients to achieve excellent compression performance by exploiting the

correlation among subbands across different decomposition levels. The modified EZW algorithm with minimum weight and difference subband approach is proposed to discard less important wavelet coefficients and retain similar visual quality as compared with the conventional EZW algorithm. Since some insignificant wavelet coefficients are eliminated, the coding gain can be improved. Experimental results show that the proposed algorithm can attain 0.5 bpp and near 1 bpp improvements as compared with the conventional EZW algorithm at the same reconstructed PSNR value using the Haar and D4 kernels respectively. The proposed algorithm also outperforms the modified EZW algorithm with minimum subband approach around 0.1 bpp at the same PSNR value using both Haar and D4 kernels. More details can be found in section 5.2.

Table of Contents

Table of Abbreviations

1-D	One Dimensional
2-D	Two Dimensional
2D-DWT	Two Dimensional – Discrete Wavelet Transform
3-D	Three Dimensional
3D-DWT	Three Dimensional – Discrete Wavelet Transform
AMRME	Adaptive Multi-resolution Motion Estimation
BMRME	Bi-directional Multi-resolution Motion Estimation
BPP	Bit Per Pixel
CIF	Common Intermediate Format
CPME-PDS	Clustered Pixel Matching Error – Partial Distortion Search
CPU	Central Processing Unit
D4	Daubechies Four
DCT	Discrete Cosine Transform
DP	Dominant Pass
DTV	Digital Television
DWT	Discrete Wavelet Transform
EMRME	Enhanced Multi-resolution Motion Estimation
FM	Frame Memory
FMRME	Fast Multi-resolution Motion Estimation
FSA	Full Search Algorithm
EZW	Embedded Zerotree Wavelet
ESA	Exhaustive Search Algorithm
GOF	Group Of Frames
HDTV	High Definition Television
HVS	Human Visual System
IZ	Isolated Zero
JPEG	Joint Photographic Experts Group
HH	High High
HL	High Low

IQ	Inverse Quantization
LH	Low High
LIS	List of Insignificant Sets
LL	Low Low
LSP	List of Significant Pixels
MC	Motion Compensation
MC-EZBC	Motion Compensated Embedded Zero Block Coding
MCTF	Motion Compensated Temporal Filtering
ME	Motion Estimation
MPEG	Motion Picture Experts Group
MRME	Multi-resolution Motion Estimation
MSE	Mean Squared Error
MV	Motion Vector
N	Negative
P	Positive
PDA	Personal Digital Assistant
PDS	Partial Distortion Search
PMA	Potential Motion Area
PSNR	Peak Signal to Noise Ratio
Q	Quantization
QCIF	Quarter Common Intermediate Format
SAD	Sum of Absolute Difference
SAQ	Successive Approximation Quantization
SEA	Successive Elimination Algorithm
SIF	Source Input Format
SP	Subordinate Pass
SPECK	Set Partition Embedded Block Coder
SPIHT	Set Partition In Hierarchical Tree
T	Zerotree Root
VLC	Variable Length Coding
ZTR	Zerotree Root

List of Figures

List of Tables

List of Publications

International Conference Papers

1. K. Y. Wong, W. C. Siu and K. C. Hui, “Fast motion estimation for wavelet-based video coding”, Proceedings of International Symposium on Intelligent Multimedia, Video and Speech Processing (ISIMP), pp. 398 – 401, Oct 2004.

Chapter 1

Introduction

1.1 Introduction of video compression

Nowadays, the usage of multimedia technologies increases exponentially, so more and more imagery and video data are manipulated. Image and video compression become a necessary procedure in the image processing and video technology for storage and transmission over network. For example, a spatial resolution of CIF (Common Intermediate Format) video sequence is 352 by 288 pixels. Each pixel is composed of three colour components, i.e., red, green and blue, for each frame and each colour component for a pixel is sampled with 8-bit precision. The storage capacity of each frame requires around 300 KBytes. The storage of a 90-minute uncoded video sequence with 30 frames per second needs approximately 50 Gbytes of digital memory. If this video sequence is transmitted at 30 frames per second without compression, the raw data rate for that video signal is greater than 13 Gbits per second. Hence, the file size of video sequence is extremely large and it is inconvenient for storage. Also, it is impossible to transmit the video signal over the network under the desirable bandwidth requirement. As a result, it is necessary to develop some video compression standards in order to achieve efficient compression for video applications.

Furthermore, the compressed video is transmitted through the communication network to great diversity of end-user requirements such as various display resolutions and decoder complexities. For example, the mobile handheld device contains the low-resolution display and low processing power, the laptop computer gives the medium-

resolution display and the high performance desktop computer attains the highest display resolution. The traditional video coding standards are difficult to achieve the multi-resolution representation for the video sequence due to the predict feedback loop used to exploit the temporal redundancy among successive video frames in the hybrid video coding model. One possible solution is to transmit more than one encoded video sequences and each compressed video sequence corresponds to a certain decoder constraint and display resolution. However, this approach consumes an erroneous amount of memory space, network bandwidth and computational effort of the video encoder since there exists lots of redundancy between encoded video streams so that it is a waste of resource to implement this approach. As a result, the scalable video compression is a new solution to solve this problem. A scalable compressed video sequence can be reconstructed by various network constraints and end-user requirements efficiently so that the encoding procedure can only be performed once and the compressed video needs not to be considered the requirements of the end-user. Only a compressed video sequence is conveyed to different end-users. In other words, the decoders with different constraints access the same encoded video sequence such that the decoder can reconstruct only a portion of the compressed video sequence according to diverse end-user requirements such as decoder complexity and resolution of display device.

In order to achieve efficient compression performance and scalability, the wavelet transform is used. Recently, the wavelet transform is used to replace the Discrete Cosine Transform (DCT), which will be mentioned in chapter two, as the transform kernel in the image processing applications. In low bit-rate applications, the DCT suffers from the blocking artifacts but the wavelet transform does not suffer from this problem according to its global decomposition in the entire image or video frame such that the

error can be distributed to the whole image or frame. Furthermore, it can provide excellent compression performance as compared with DCT. Besides, the wavelet transform is scalable in nature so that it can offer the multi-resolution representation of the image in order to fulfill the requirements of great diversity of end-users. The latest image compression standard, JPEG2000, makes use of the wavelet transform as the transform kernel in order to achieve both superior compression efficiency and scalability as compared with the conventional image-coding standard, JPEG. However, the wavelet transform employed in the video coding is still an open research area. Many researchers are still putting effort in this field to achieve full scalability, such as temporal and spatial scalabilities, in the video sequence.

1.2 Organization of Thesis

This thesis is organized in six chapters. Chapter two reviews some basic concepts of the conventional video-coding standards, the traditional algorithms of motion estimation in the spatial and wavelet domains, the fundamental idea of wavelet transform, a classical approach to encode the wavelet coefficients (Embedded Zerotree Wavelet algorithm) and the framework of motion compensated temporal filtering (MCTF) used in the three dimensional discrete wavelet transform (3D-DWT) video coder. In chapter three, a new motion estimation algorithm in the wavelet domain in order to improve the speed of motion estimation in the wavelet domain by making use of the correlation among corresponding subbands at different decomposition levels in the wavelet pyramid will be discussed in detail. Chapter four presents a fast motion estimation scheme used in the 3D-DWT video coder by exploiting the relationship of the wavelet-transformed frames at different temporal decomposition levels. In chapter five, a modified EZW algorithm will be proposed to

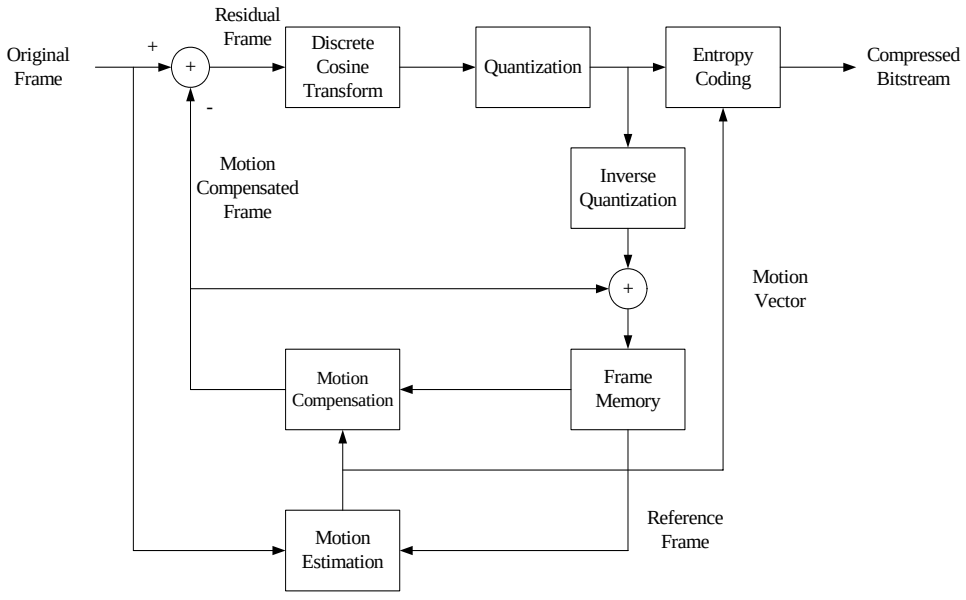
improve the compression performance by removing less important information while retaining the same visual quality as the conventional EZW algorithm. Finally, a conclusion of this thesis will be drawn and some possible future directions of this study will be given in chapter six.

Chapter 2

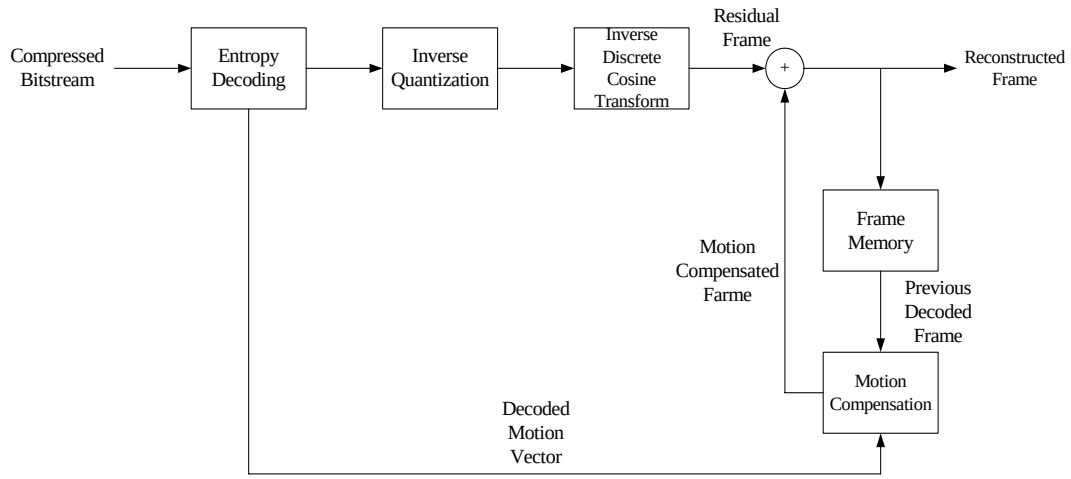
Technical Review

2.1 Hybrid video coding model

Modern video coding standards, such as the MPEG-1 [1], MPEG-2 [2], MPEG-4 [3] and H.263 [4], can achieve high compression efficiency with different applications by using the hybrid video coding model as depicted in Figure 2.1 to remove both spatial and temporal redundancies between successive frames. Figures 2.1 (a) and (b) illustrate a simplified block diagram of the conventional hybrid video encoder and decoder respectively. The functional blocks are the tasks that are usually used by various video coding standards.



(a)



(b)

Figure 2.1 Simplified block diagram of the hybrid video (a) encoder and (b) decoder

Motion estimation is used to reduce the temporal redundancy between consecutive video frames. The input video frame, i.e. the current frame, is divided into many non-overlapping blocks and each block finds its best match location in the reference frame, which is the previously encoded frame. According to many experimental results, most motion activities in the natural video sequence is purely translational, i.e. an object is moving in translational motion across a nearly stationary background without rotation, expansion or dilation, so this is an important assumption that there only exists the purely translational motion in the motion model used in the motion estimation. After finding the best match position in the reference frame, the motion information or motion vector, is encoded instead of coding the pixels in the current input frame. After that, the predicted frame or motion-compensated frame is constructed by taking relevant blocks from the reference frame according to the motion vector information found by motion estimation, and this process refers to as motion compensation. The motion-compensated frame is only a prediction of the current frame so that there may exist some differences between the current and motion-compensated frames. Therefore, a residual frame is formed by subtracting the current frame from the motion-compensated frame, and it is also encoded. As a result, the interframe

redundancy can be removed by motion estimation and motion compensation. Furthermore, there may exist some correlations between the neighbouring pixels in the residual frame so that the Discrete Cosine Transform (DCT) is used to remove the remaining spatial correlation inside the residual frame during interframe mode coding.

The DCT is responsible for removing the intraframe redundancy, i.e. the spatial correlation, within a frame. It can be used in both interframe and intraframe modes coding. The intraframe mode coding refers to the coding of still image just similar to current image coding standards, such as JPEG, in order to achieve compression. An intraframe refers to the case without using any reference to the previously encoded frame and it can only exploit to the spatial correlation. On the contrary, the DCT can also be used to remove the relationship between the pixels inside the motion-compensated frame, i.e. prediction error in the motion-compensated prediction, during interframe mode coding. An interframe refers to using the previously encoded frame, which can be the previous or future frames or both, to achieve compression and it can make use of both properties of spatial and temporal correlations. The DCT is just a mapping from the time domain to the frequency domain such that most energy is concentrated to a few portions of the transform coefficients and the transform coefficients are entropy encoded in order to achieve good compression performance. Since this codec is designed for eliminating both spatial and temporal redundancies, so it is called hybrid video coding.

Quantization is the next step of the Discrete Cosine Transform. It is used to remove the less important transform coefficients by reducing the precision of the coefficients such that the bit rate can be reduced and the acceptable visual quality is preserved. After that, the quantized DCT coefficients and motion information obtained by motion estimation are entropy encoded by variable length coding such as the

Huffman coding which assigns a shorter codeword to frequently appeared data or vice versa such that the compressed bitstream can be formed. During decoding, the bitstream is received and decoded by the look-up table which is the same table as the encoder side in order to preserve the original information.

The motion estimation and compensation using the traditional block-based model will be discussed in section 2.2. The classic motion estimation algorithm, Full Search Algorithm (FSA), and fast motion estimation algorithms will be mentioned in sections 2.2.1 and 2.2.2 respectively. In recent years, the wavelet transform is widely used in image processing and video technology in order to improve the coding gain. The wavelet transform will be reviewed in section 2.3. The Embedded Zerotree Wavelet (EZW) algorithm is one of the typical methods for coding the wavelet coefficients and it will be studied in section 2.4. Some researchers enhanced the coding efficiency of the EZW algorithm and the modifications of EZW algorithm will be mentioned in section 2.5. Besides the EZW algorithm, the Set-Partition Embedded Block Coder (SPECK) algorithm receives much attention due to its superior coding performance as compared with the EZW algorithm. It will be discussed in section 2.6. The two dimensional discrete wavelet transform is applied in video technology and some motion estimation and compensation algorithms in wavelet domain are developed. They will be analyzed in sections 2.7 and 2.8 respectively. In recent years, the three dimensional discrete wavelet transform is developed to further enhance the coding efficiency in video coder and it will be discussed in sections 2.9, 2.10 and 2.11 respectively. Finally, a concluding remark of this chapter will be briefly drawn in section 2.12.

2.2 Block-based motion estimation and compensation

The block-based motion estimation and compensation are widely used in modern video coding system for compression and the block-based model is effective to deal with the video sequence with translational motion as discussed in the previous section. The objective of motion estimation and compensation is to eliminate the temporal redundancy among successive video frames since there exists a huge correlation between consecutive frames in order to achieve compression. During motion estimation, the current frame is divided into many non-overlapping regions which are called blocks as depicted in Figure 2.2. The values of pixels in this current frame are predicted from the pixels of another block in the reference frame, which is previously encoded, in a certain search range. The reference block is the best-matched block of the current block to give the minimum difference between reference and current blocks. As illustrated in Figure 2.2, the motion vector represents the displacement between these two blocks and is only encoded instead of the pixel value of the block in the current frame. As a result, the motion estimation is carried out to obtain the motion vectors. After finding the motion vector, the reference frame is to work with it to produce the predicted frame, i.e. the motion-compensated frame, which is the prediction of the current frame from the motion compensation process. The residual frame, i.e. the current frame subtracting to the motion-compensated frame, and the motion vectors are sent to the decoder for reconstruction of the current frame.

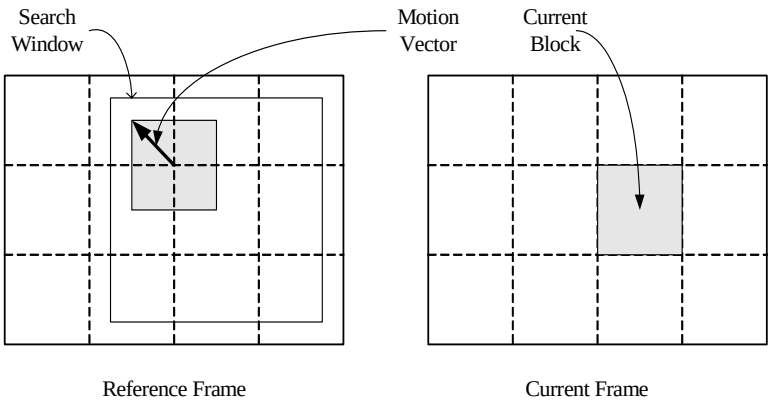


Figure 2.2 The illustration of finding a motion vector

The motion estimation finds the best matched location of the current frame within a certain search range of the reference frame by minimizing the distortion between the target block in the current frame and a candidate block inside the search window of the reference frame. There are two criteria, which are the Sum of the Absolute Difference (SAD) and the Mean Squared Error (MSE), that are most frequently used in the motion estimation process. The SAD between a target block at position *(x, y)* in the current frame, I_t, and a candidate block at position *(x+u, y+v)*, in the reference frame, I_{t-1}, is shown in equation 2.1.

$$SAD(x,y;u,v)=\sum_{j=0}^{N-1}\sum_{i=0}^{N-1}\left|I_t(x+i,y+j)-I_{t-1}(x+i+u,y+j+v)\right| \tag{2.1}$$

where $N \times N$ is the size of the block, $I_t(x, y)$ and $I_{t-1}(x, y)$ represent the pixel intensities at the initial position *(x, y)* of the target block of the current frame and candidate block of the reference frame respectively, *(i, j)* is used to locate the position of a pixel inside a block and *(u, v)* is the location of a possible motion vector. The SAD indicates the absolute difference between the target block in the current frame and the candidate block in the reference frame by pixel-by-pixel computation. The MSE is another distortion measure between the target block in the current frame and the candidate block in the reference as shown in equation 2.2.

$$MSE(x,y;u,v)=\frac{1}{N^2}\sum_{j=0}^{N-1}\sum_{i=0}^{N-1}\left[I_t(x+i,y+j)-I_{t-1}(x+i+u,y+j+v)\right]^2 \quad (2.2)$$

The MSE is similar to the SAD but it involves the multiplication and division so that the computation complexity is higher than that of the SAD and its performance is comparable to that of the SAD. As a result, the SAD is widely used for the distortion measure during motion estimation. After finding the SAD values of all possible locations within the search window of the reference frame, the location with the minimum error indicates the desired motion vector which represents the location of the best match block in the reference frame giving the minimum error. The motion vector is defined in equation 2.3.

$$(u_x,v_y)=\arg\min_{(u,v)\in W} SAD(x,y;u,v) \quad (2.3)$$

where $W=\{(u,v)\,|\,-M\le u,v\le M-1\}$ is a set of all possible locations in the search window as depicted in Figure 2.2 and *M* is the possible maximum displacement of the motion vector, *(u, v)*. The motion vector specifies the location that a block in the reference frame is copied to reconstruct the current frame during decoding. Thus, it is also encoded and transmitted to the decoder in order to reconstruct the current frame. If the motion vector can accurately represent the motion in the video sequence, the number of bits used to encode the residual frame (error frame) will become smaller due to small errors in the residual frame or vice versa.

A classic searching algorithm, full search algorithm (FSA), is discussed in the section 2.2.1. However, its computational complexity is extremely high, so some fast searching algorithms for lossless approach in order to improve searching speed are reviewed in section 2.2.2.

2.2.1 Full Search Algorithm (FSA)

The full search algorithm (FSA) or exhaustive search algorithm (ESA) is used to find out the optimal motion vector by testing all possible locations inside the search window. Since all possible search locations have been checked, the global minimum matching error of the obtained motion vector pointing to the candidate block in the reference frame can be guaranteed. Although the global minimum can be assured, its computational complexity is extremely high owing to testing all possible locations within the search window of the reference frame. It consumes most of the computational effort of the hybrid video coding during encoding of a video sequence. Consequently, there are some fast algorithms that can reduce the computational burden of the motion estimation in a lossless way introduced in section 2.2.2. The lossless approach refers to that the obtained motion vectors of a fast searching algorithm are exactly the same as those to be obtained from the FSA.

2.2.2 Fast Searching Algorithms for lossless approach

The computational effort of FSA for motion estimation is enormously huge so that some fast algorithms are developed to reduce the number of operations used in motion estimation. In this section, the lossless and fast motion estimation algorithms are reviewed such that the motion vectors obtained by these fast lossless methods are the same as that of the FSA.

2.2.2.1 Partial Distortion Search (PDS)

The partial distortion search (PDS) algorithm [12] can be characterized as the fast full search algorithm (FSA) since it can attain the same set of motion vector fields

as that of FSA but its computational complexity is much fewer than that of the FSA. The philosophy behind the PDS is that after the SAD value of first block in the reference frame and the target block in the current frame is calculated, this value is stored. It is then considered as the temporary minimum SAD error. After that, the SAD value of the first row in other location is calculated, its partial SAD compared to the temporary minimum error. If this accumulated error is already greater than temporary minimum error, the SAD calculation of the remaining rows is terminated since the partial SAD value of this candidate block is already greater than the temporary minimum error. Otherwise, if the SAD value of this candidate block is smaller than the temporary minimum error, this new minimum error will become the temporary minimum error to be used comparing the remaining locations within the search window. Then, this procedure is repeated again until all possible locations inside the search window are tested. The partial SAD value is given as below.

$$SAD_p(u, y, u_x, v_y) = \sum_{i=0}^{p} \sum_{j=0}^{N-1} \left| I_t(x+i, y+j) - I_{t-1}(x+u+i, y+v+j) \right| \tag{2.4}$$

where p is the number of row for which the accumulation of errors is terminated and $p = 0, 1, 2, \ldots, N-1, N$ is the size of a block, (x, y) is the location of a block in the current frame, (i, j) represents the pixel inside a block, (u, v) indicates the possible motion vector and I_t and I_{t-1} identify the pixel intensities at certain position within the current and reference frames respectively. The second summation stands for the sum of absolute difference between the target block in the current frame and the candidate block in the reference frame while the first summation denotes the number of rows to terminate the SAD calculation inside a candidate block. The SAD calculation is terminated with a value of p which is equal to any integer value between 0 and $N-1$. If $SAD_p(x, y, u, v) \geq SAD_p(x, y, u_{temp}, v_{temp})$, the calculation of SAD value of the whole

block is terminated as this position cannot give the lowest error, where (u_{temp}, v_{temp}) specifies the motion vector that accounts for the lowest SAD value. When the value of *p* is small, the process of SAD calculation can be terminated early.

For most video sequences, their frame rate is between 15 to 30 frames per second and 30 to 60 frames per second for low and high quality applications respectively so that the time interval between each frame is very small. As a result, the movement of a block in the video sequence usually varies slowly leading to the magnitudes of the motion vectors are also very small especially for low bit-rate video applications. Consequently, the magnitudes of most motion vectors are very close to zero. Usually, about 80% of the blocks can be regarded as stationary or quasi-stationary (enclosed in the central 3×3 area) and most of the motion vectors are enclosed in the central 5×5 area. According to this observation [38], the motion vector distribution of the natural video sequence is highly centre-biased. Due to this particular form of distribution, the searching pattern within the search window should be centre-biased as illustrated in Figure 2.3. The outward spiral searching strategy is usually adopted in the PDS in order to terminate the SAD calculation as soon as possible so that the optimal motion vectors can be found quickly.

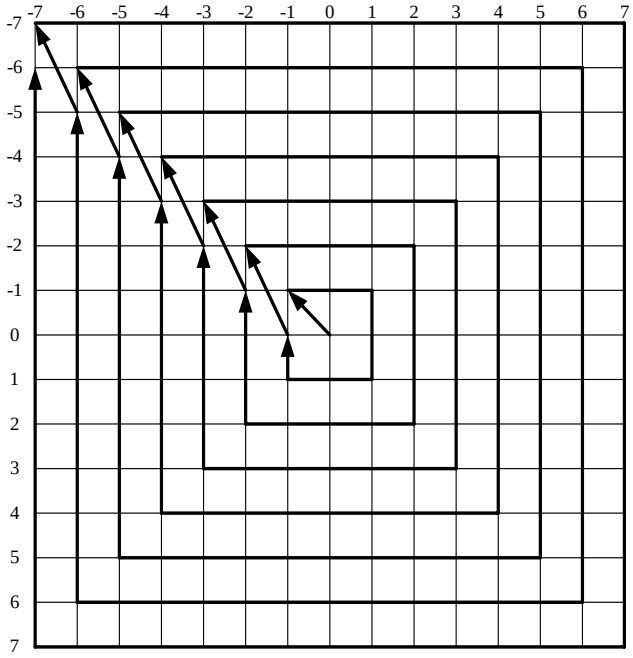


Figure 2.3 Spiral searching path within a search window with the size of ±7

2.2.2.2 Clustered Pixel Matching Error - Partial Distortion Search (CPME-PDS)

The previous section described a fast and lossless algorithm, the PDS, to get the motion vectors. The partial SAD is calculated by adding the absolute difference between the target block in the current frame and the candidate block in the current frame in a row by row and from top to bottom computation. If the partial SAD value of the first row is just greater than the temporary minimum error, then the SAD calculation of that candidate block is stopped. Otherwise, the accumulation of error calculation continues. However, the pixels, which give the greatest error, may not be located in the first row. They may be in the last row and any one of the rows inside the candidate and target blocks. Therefore, if we find out the pixels that provide the largest error and use them to calculate the partial SAD value, the speed of motion estimation can be further enhanced. Hence, it is necessary to investigate the spatial distributions of pixel matching error inside a block. A phenomenon reported in the literature says that the

pixels with similar matching errors in magnitude tend to appear together in clusters [18]. According to this observation, the Clustered Pixel Matching Error for Partial Distortion Search (CPME-PDS) [18] has been suggested to improve the speed of motion estimation. The problem is how to find the pixel that provides the largest matching error in order to terminate a partial SAD calculation.

For a target block, the positions of its pixels are represented by an index set, $S = \{(k_n, l_n) | n = 0,..., N-1\}$, where *k* and *l* are the coordinates of a pixel and *N* is the number of pixels in a block. For a single pixel at $s_n = (k_n, l_n), s_n \in S$, its matching error is $e(s_n) = I_t(s_n) - R(s_n)$, where $R(s_n)$ is a random variable which represents the pixel value at s_n of a candidate block. For the sake of simplicity, in the following discussion, s_n is replaced by *n*, and both block location (*x, y*) and motion vector (*u, v*) are dropped. To improve the saving in computation of a PDS, pixel matching errors with an ideal index set much follow this rule, $e(0)^2 \geq e(1)^2 \geq ... \geq e(n)^2 \geq ... \geq e(N-1)^2$. The predicted pixel matching errors, $p(n)$, is defined as, $p(n) = I_t(n) - m$, where *m* is a reference value to be used to obtain the prediction. The value of *m* is obtained by minimizing the expected value of the sum of squares of the difference between $e(n)^2$ and $p(n)^2$,

i.e.
$$m_{opt} = \arg\min_m \left\{ E\left[\sum_{n=0}^{N-1} \left[e(n)^2 - p(n)^2 \right]^2 \right] \right\}$$

$$m_{opt} = \arg\min_m \left\{ E\left[\sum_{n=0}^{N-1} \left[[I_t(n) - R(n)]^2 - [I_t(n) - m]^2 \right]^2 \right] \right\}$$

In order to find *m*, the above equation is differentiated respect to *m*, then we can obtain

$$\frac{d}{dm} \left\{ E\left[\sum_{n=0}^{N-1} \left[[I_t(n) - R(n)]^2 - [I_t(n) - m]^2 \right]^2 \right] \right\} = 0$$

By substituting $R(n) = I_t(n) - e(n)$ into the above equation, it becomes

$$\frac{d}{dm}\left\{E\left[\sum_{n=0}^{N-1}\left[e(n)^2-\left[I_t(n)-m\right]^2\right]^2\right]\right\}=0$$

$$E\left[-\sum_{n=0}^{N-1}4\left[e(n)^2-\left[I_t(n)-m\right]^2\right]\left[I_t(n)-m\right]\right]=0$$

$$E\left[-\sum_{n=0}^{N-1}\left[e(n)^2 I_t(n)-e(n)^2 m-\left[I_t(n)-m\right]^3\right]\right]=0$$

$$E\left[-\sum_{n=0}^{N-1}\left[e(n)^2 I_t(n)-e(n)^2 m-\left[I_t(n)-m\right]^3\right]\right]=0$$

$$E\left[-\sum_{n=0}^{N-1}\left[e(n)^2 I_t(n)-e(n)^2 m-\left[I_t(n)^3-mI_t(n)^2-2mI_t(n)^2+2m^2 I_t(n)+m^2 I_t(n)-m^3\right]\right]\right]=0$$

$$E\left[-\sum_{n=0}^{N-1}\left[e(n)^2 I_t(n)-e(n)^2 m-I_t(n)^3+3mI_t(n)^2-3m^2 I_t(n)+m^3\right]\right]=0$$

$$E\left[-\sum_{n=0}^{N-1}\left[m^3-3I_t(n)m^2+\left(3I_t(n)^2-e(n)^2\right)m+I_t(n)e(n)^2-I_t(n)^3\right]\right]=0$$

Finally, the following cubic equation can be obtained,

$$\therefore \qquad m^3-3\overline{I_t}m^2+\left(3\overline{I_t^{\,2}}-\overline{e^2}\right)m+\overline{I_t e^2}-\overline{I_t^{\,3}}=0,$$

where $\overline{e^2}=\frac{1}{N}E\left[\sum_{n=0}^{N-1}e(n)^2\right]$, $\overline{I_t e^2}=\frac{1}{N}E\left[\sum_{n=0}^{N-1}I_t(n)e(n)^2\right]$, $\overline{I_t}=\frac{1}{N}E\left[\sum_{n=0}^{N-1}I_t(n)\right]$,

$$\overline{I_t^{\,2}}=\frac{1}{N}E\left[\sum_{n=0}^{N-1}I_t(n)^2\right] \text{ and } \overline{I_t^{\,3}}=\frac{1}{N}E\left[\sum_{n=0}^{N-1}I_t(n)^3\right]$$

Due to the assumption that the natural image is dominated by low frequency components, three approximate real roots *m* of the above cubic equation can be obtained,

$$m\approx\overline{I_t} \qquad \text{or} \qquad m\approx\overline{I_t}\pm\sqrt{\overline{e^2}}$$

The first approximated root is the mean of pixel values in the target block. For the second root, *e(n)* often consists of two components, $e_m(n)$ and $e_w(n)$, where $e_w(n)$ denotes zero-mean white noise with negligible magnitudes and $e_m(n)$ represents errors

due to irregular motions, light variation, etc. They tend to have the same sign in a block. Hence, the expected value of the random variable representing the pixel value at *n* of a candidate block is, $\overline{R} \approx \overline{I_t} \pm \left|\overline{e}\right| \approx \overline{I_t} \pm \sqrt{\overline{e^2}}$, which is the approximation of *m* . Therefore, the first approximated real root, the mean of pixel values in a candidate block, is determined as the root of the above cubic equation.

The objective of the CPME-PDS is that the PDS can terminate a partial SAD calculation to reject the candidate block earlier when the location of global minimum matching error is met in a search earlier. To achieve this purpose, two strategies are used as shown below.

1. The outward spiral scanning is used to exploit the center-biased motion vector distribution characteristics of the real world video sequence [38].
2. The correlation in the motion field is exploited by using a median predictor of three adjacent blocks, left, top and top right blocks to the current position as the initial searching point of the spiral scanning as illustrated in Figure 2.4. The median predictor has been used as described in [3].

According to the above considerations and analytical results, the mean of pixel values in the candidate block of the initial searching point is used to compute the reference value, *m*, because we can assume that

$$\overline{I_{t-1}(i+u_{med},\, j+v_{med})} \approx \overline{I_t(i,\, j)}$$

where (u_{med}, v_{med}) = the median predictor which is illustrated in Figure 2.4. The expected pixel matching error, $p_{exp}(n)$, of each pixel in the target block is calculated with *m*. The required adaptive index set, *S*, is given by sorting $\left|p_{exp}(n)\right|$ in descending order. The partial SAD value is calculated with *S* during the searching in an outward spiral scanning. The CPME-PDS approach can be summarized as follows:

Step 1) Determine the median predictor, (u_{med}, v_{med}), of the three adjacent blocks.

Step 2) Calculate the reference value, *m*, with the median predictor, (u_{med}, v_{med}), where the block size is 15×15.

$$m = \frac{1}{256}\sum_{j=0}^{15}\sum_{i=0}^{15} I_{t-1}(x + u_{med} + i, y + v_{med} + j)$$

Step 3) Initialize an index set, $S' = \{(k'_n, l'_n) | n = 0, ..., N-1\}$, which represents all pixels of the target block, where *k'* and *l'* are the coordinates of a pixel and *N* is the number of pixels in a block.

Step 4) Calculate the expected absolute pixel matching error, $|p_{exp}(n)|$, of each pixel in the target block.

$$|p_{exp}(n)| = |I_t(k'_n, l'_n) - m|$$

Step 5) Rearrange the order of set S' to obtain an adaptive index set *S* by sorting the expected absolute pixel matching error, $|p_{exp}(n)|$, in descending order, such that, $S = \{(k_n, l_n) | n = 0, ..., N-1\}$. The $p_{exp}(n)$ corresponding to the order of the sorted index set, *S*, has the following feature,

$$|p_{\exp}(0)| \geq \ldots \geq |p_{\exp}(n)| \geq \ldots \geq |p_{\exp}(N-1)|$$

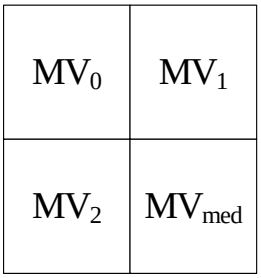
Step 6) Apply the adaptive index set, *S*, to calculate the partial SAD value during the searching in an outward spiral scanning.

Note that the adaptive index set is established on a pixel-based approach, so this algorithm is entitled as the pixel-based CPME-PDS. According to the pipeline structure of the Central Processing Unit (CPU), the time used to access the memory of the pixel values in random location is much longer than that used to access the memory in a row of pixel values within a block. Although the number of operations is reduced dramatically, the execution time is longer as compared with the PDS due to the memory

access problem in CPU. In order to overcome this difficulty, the row-based CPME-PDS can be used. The pixels are sorted row by row in a block for SAD_p accumulation in order to make use of the advantage of clustering characteristic. By using an identical reference value, *m*, the expected absolute pixel error of a row of 16 pixels is calculated as follows, which is used to determine the accumulating order.

$$p_{\exp} = \sum_{x=0}^{15} \left| I_t(x, l'_n) - m \right|$$

Even though the number of operations used in row-based CPME-PDS is higher than that of the pixel-based CPME-PDS, its execution time is shorter than that of pixel-based approach during implementation. Also, both number of operations and execution time of the row-based approach are reduced as compared with the PDS by making use of the clustering characteristics of pixels.



$MV_{med}(x) = \text{median}\ (MV_0(x), MV_1(x), MV_2(x))$
$MV_{med}(y) = \text{median}\ (MV_0(y), MV_1(y), MV_2(y))$

Figure 2.4 Median predictor of three adjacent blocks, top right, top and left blocks to the current block

2.2.2.3 Successive Elimination Algorithm

The previous two sections discussed the PDS algorithm to reject the impossible candidate blocks earlier by terminating the partial SAD calculation as soon as possible in all possible searching locations within the search window of the reference frame. If only a part of the searching position is tested, the computational burden of motion estimation can be further reduced. A fast searching technique called Successive

Elimination Algorithm (SEA) makes use of this method to find out the motion vectors and the accuracy of the motion vectors is the same as that of FSA. The concept of SEA is that if a candidate block in the reference frame can meet certain requirement, then a further checking of this candidate block is performed. Otherwise, it is rejected and another candidate block along the spiral-scanning path is tested until all possible locations inside the search window are exhausted. The idea of SEA comes from this inequality,

$$\big||a|-|b|\big| \le |a-b| \tag{2.4}$$

where *a* and *b* are real numbers. This inequality means that the absolute value of the difference between two real numbers is always larger than or equal to the absolute value of the difference between the absolute values of the individual numbers. For example, if a and b are the +ve and –ve real numbers respectively, the right hand side (RHS) must be greater than the left hand side (LHS) of the above inequality. Let us examine the inequality 2.4 carefully, it implies the inequality equivalent to the inequalities 2.5 because the absolute difference of two absolute numbers, $|a|$ and $|b|$, say for example can be formed either by subtracting $|a|$ from $|b|$ or $|b|$ form $|a|$.

$$\left.\begin{aligned} |a|-|b| &\le |a-b| \\ |b|-|a| &\le |a-b| \end{aligned}\right\} \tag{2.5}$$

The temporary SAD value, $SAD_{temp}\left(x, y; u_{temp}, v_{temp}\right)$, in the PDS algorithm is defined in equation 2.6.

$$SAD_{temp}\left(x, y; u_{temp}, v_{temp}\right) = \sum_{j=0}^{N-1}\sum_{i=0}^{N-1}\left|I_t\left(x+i, y+j\right) - I_{t-1}\left(x+i+u_{temp}, y+j+v_{temp}\right)\right| \tag{2.6}$$

where (u_{temp}, v_{temp}) is the temporary motion vector, $N \times N$ is the size of block, $I_t(x, y)$ and $I_{t-1}(x, y)$ represent the pixel intensities at the initial position (x, y) of the target block of the current frame and candidate block of the reference frame respectively and (i, j) is used to locate the position of a pixel inside a block. The objective of motion estimation

is to find out a better candidate block such that $SAD(x,y;u,v) \le SAD_{temp}(x,y;u_{temp},v_{temp})$, where $SAD(x,y;u,v)$ is the possible minimum SAD value and *(u, v)* is the possible optimal motion vector. The difference term of the SAD value, $|I_t(x+i,y+j) - I_{t-1}(x+i+u,y+j+v)|$, is substituted into the inequalities 2.5 and then we have,

$$|I_t(x+i,y+j)| - |I_{t-1}(x+i+u,y+j+v)| \le |I_t(x+i,y+j) - I_{t-1}(x+i+u,y+j+v)| \quad (2.7)$$

$$|I_{t-1}(x+i+u,y+j+v)| - |I_t(x+i,y+j)| \le |I_t(x+i,y+j) - I_{t-1}(x+i+u,y+j+v)| \quad (2.8)$$

If the inequalities 2.7 and 2.8 perform summation in both sides for all pixels in a block, they will become the inequalities 2.9 and 2.10 as shown below.

$$\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}|I_t(x+i,y+j)| - \sum_{i=0}^{N-1}\sum_{j=0}^{N-1}|I_{t-1}(x+i+u,y+j+v)| \le \sum_{i=0}^{N-1}\sum_{j=0}^{N-1}|I_t(x+i,y+j) - I_{t-1}(x+i+u,y+j+v)| \quad (2.9)$$

$$\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}|I_{t-1}(x+i+u,y+j+v)| - \sum_{i=0}^{N-1}\sum_{j=0}^{N-1}|I_t(x+i,y+j)| \le \sum_{i=0}^{N-1}\sum_{j=0}^{N-1}|I_t(x+i,y+j) - I_{t-1}(x+i+u,y+j+v)| \quad (2.10)$$

The first summation in the LHS of inequality 2.7 represents the sum norm of the target block in the current frame denoted by *R(x, y)* and the second one indicates the sum norm of any matching candidate block in the reference frame with motion vector *(u, v)* and is denoted by *M(x, y; u, v)*. The summation in the RHS of inequality 2.7 specifies the SAD value corresponding to the motion vector *(u, v)* denoted by *SAD(x, y; u, v)*. Thus, the inequalities 2.7 and 2.8 can be rewritten as follows.

$$R(x,y) - M(x,y;u,v) \le SAD(x,y;u,v) \quad (2.11)$$

$$M(x,y;u,v) - R(x,y) \le SAD(x,y;u,v) \quad (2.12)$$

Say for example, the temporary SAD value, $SAD_{temp}(x, y; u_{temp}, v_{temp})$, of a candidate block with motion vector (u_{temp}, v_{temp}) is already a good matching block corresponding to the target block with the temporary minimum SAD value. Then, we are looking for a better matching candidate block with motion vector *(u, v)* such that,

$$SAD(x,y;u,v) \le SAD_{temp}(x,y;u_{temp},v_{temp}) \quad (2.13)$$

After that, we relate the inequality 2.13 with inequalities 2.11 and 2.12 which become,

$$R(x,y) - M(x,y;u,v) \le SAD(x,y;u,v) \le SAD_{temp}(x,y;u_{temp},v_{temp}) \quad (2.14)$$

$$M(x,y;u,v) - R(x,y) \le SAD(x,y;u,v) \le SAD_{temp}(x,y;u_{temp},v_{temp}) \quad (2.15)$$

The inequalities 2.14 and 2.15 can be written as,

$$R(x,y) - M(x,y;u,v) \le SAD_{temp}(x,y;u_{temp},v_{temp}) \quad (2.14)$$

$$M(x,y;u,v) - R(x,y) \le SAD_{temp}(x,y;u_{temp},v_{temp}) \quad (2.15)$$

which implies,

$$R(x,y) - SAD_{temp}(x,y;u_{temp},v_{temp}) \le M(x,y;u,v) \le R(x,y) + SAD_{temp}(x,y;u_{temp},v_{temp}) \quad (2.16)$$

The inequality 2.16 is the major result of SEA. If the range of the value of sum norm of any candidate block in the reference frame can meet the requirement of the above inequality, then a further checking using PDS, say for example, is performed. Otherwise, no further checking is required since the SAD value of this candidate block must be greater than the temporary SAD value, $SAD_{temp}(x, y; u_{temp}, v_{temp})$, which has already obtained before. Furthermore, since some impossible candidate blocks are rejected and not required to perform searching, so the computational complexity can be reduced significantly without excluding the optimum block because the blocks that can satisfy the inequality 2.16 must be fewer than all blocks in the search window.

During the implementation of inequality 2.16, the sum norm of each block inside the search window has to be known before. A fast method is used to calculate the sum norm. Suppose that the frame size is $H \times W$. The whole image is divided into $(H-N+1)$ row strips and each row strip contains N rows as depicted in Figure 2.5.

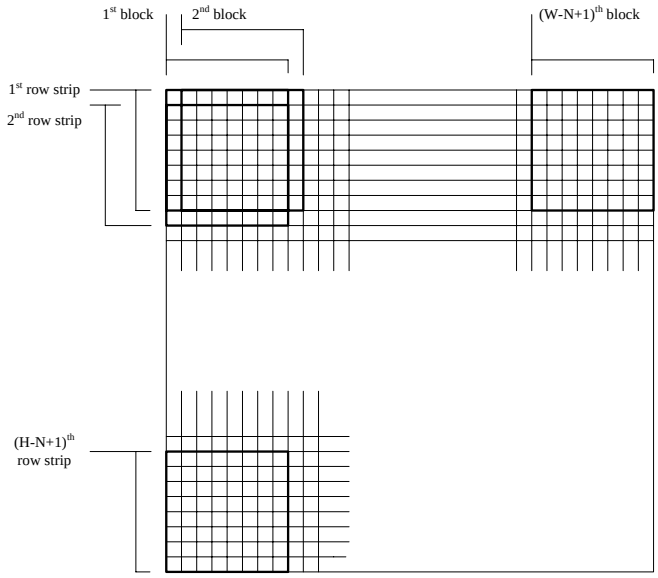


Figure 2.5 Geometry for the calculation of the sum norm

Firstly, we calculate the sum of each column and save them as C_{11}, C_{12}, …, C_{1W} for the first row strip. For the second row strip, $C_{21} = C_{11} - I(1,1) + I(N+1,1)$, $C_{22} = C_{12} - I(1,2) + I(N+1,2)$, …, $C_{2W} = C_{1W} - I(1,W) + I(N+1,W)$. Similarly, the sum of each column for other row strips is calculated in this way.

Secondly, the sum norm of the first block denoted by SN_{11} is calculated by adding C_{11}, C_{12}, …, C_{1N}. As the block is shifted by one pixel horizontally, so the sum norm of the second block denoted by SN_{12} is $SN_{11} - C_{11} + C_{1(N+1)}$. Similarly, the other sum norms, SN_{13}, SN_{14}, …, $SN_{1(W-N)}$ can be found and the sum norm of each block in the following row strips can be obtained in the same way.

The computation of getting the sum norm can be considered as the search overhead but this computational effort is very small. With this small overhead, the total number of operations for motion estimation can be considerably reduced.

2.3 The Wavelet Transform

Recently, the wavelet transform receives much attention as an alternative of the Discrete Cosine Transform (DCT). Similar to the DCT, it transforms the signal into another domain such that both time and frequency information can be preserved. Due to its superior performance in compression efficiency as compared with the DCT and scalability in nature, it is widely used in various applications such as image and video compression, image retrieval and de-noising. For the image coding, the latest image coding standard, JPEG 2000 [47], [48], makes use of the Discrete Wavelet Transform (DWT) as the transform kernel instead of the DCT which is used in previous image coding standard, JPEG. In the following, the basic knowledge of wavelet transform and the lifting realization of the wavelet transform will be reviewed.

2.3.1 Brief introduction of the wavelet transform

The wavelet transform is widely used in various applications such as image processing and video technology. It acts as a transform kernel to map the pixels of an image or video frame in the spatial domain to the wavelet coefficients in the wavelet domain and can preserve both time and frequency information. The decomposition and reconstruction of two-band structure of one-dimensional wavelet transform with one decomposition level is illustrated in Figure 2.6. The input signal is carried out low-pass and high-pass filtering in the analysis stage. Then, both filtered signals are downsampled by a factor of two in order to maintain the same size of samples before decomposition in order to extract the low and high frequency parts of original signal. In the synthesis stage, the reverse operations are executed in order to obtain back the reconstructed signal. If the reconstructed signal is exactly the same as the original signal,

this system is said to be “perfect reconstruction (PR)”. For images, the two-dimensional Discrete Wavelet Transform (2D-DWT) is applied. Figure 2.7 shows the decomposition of the 2D-DWT with three decomposition levels. After decomposing the image into four subbands, LL, LH, HL and HH, in the first level, only the LL subband is further decomposited in the remaining decomposition levels. For example, if three levels are decomposited, then ten subbands are generated as shown in Figure 2.8(a). The transformed version of “Fruit” image in three decomposition levels using the Daubechies-4 kernel is depicted in Figure 2.8(b).

Unlike the DCT, the wavelet transform does not suffer from the blocking artifacts in low bit-rate applications. Besides, it can provide a multiresolution representation of the original image. For example, the low-resolution image is obtained by discarding the high frequency subbands. The LL subband in Figure 2.8(b) yields a high quality of the low-resolution version of the original image.

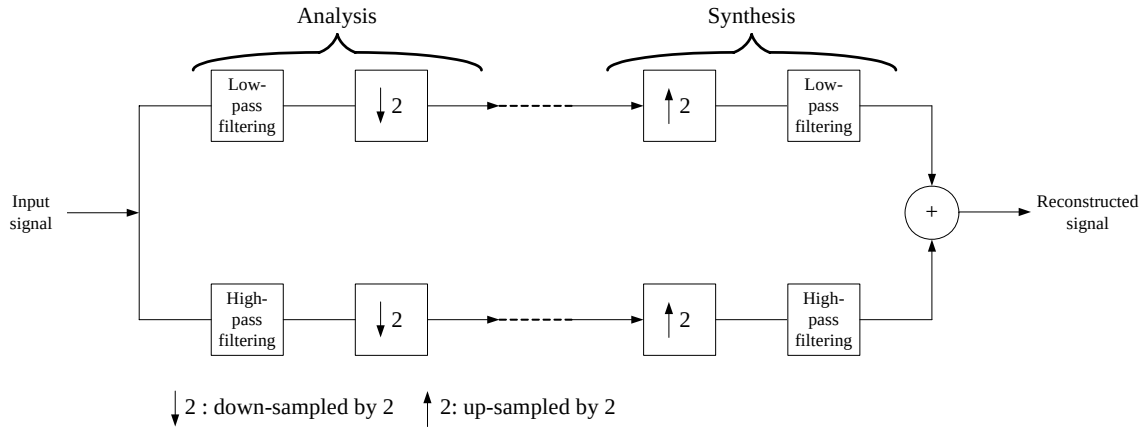


Figure 2.6 The block diagram of analysis and synthesis of one-dimensional wavelet transform with one decomposition level

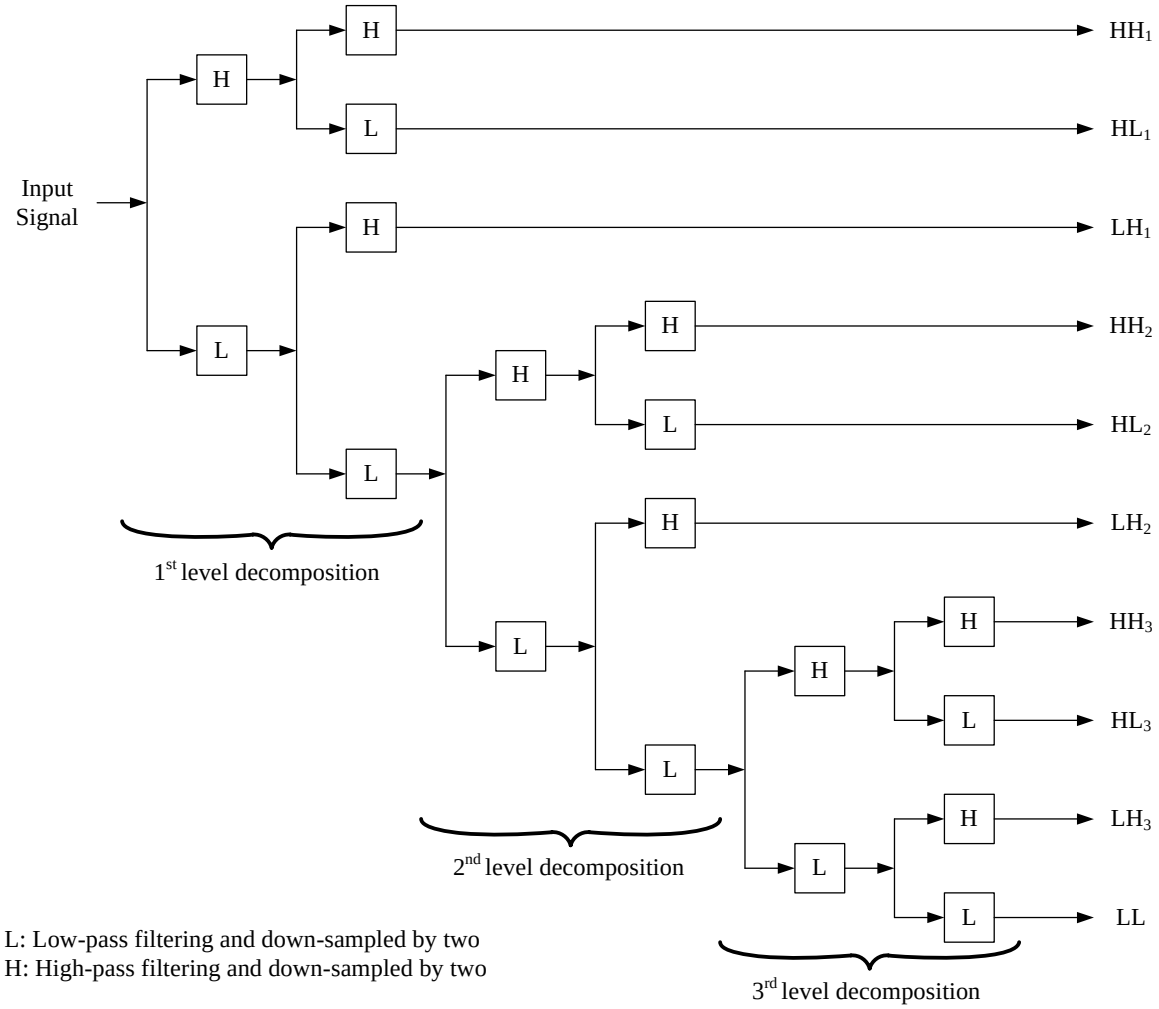
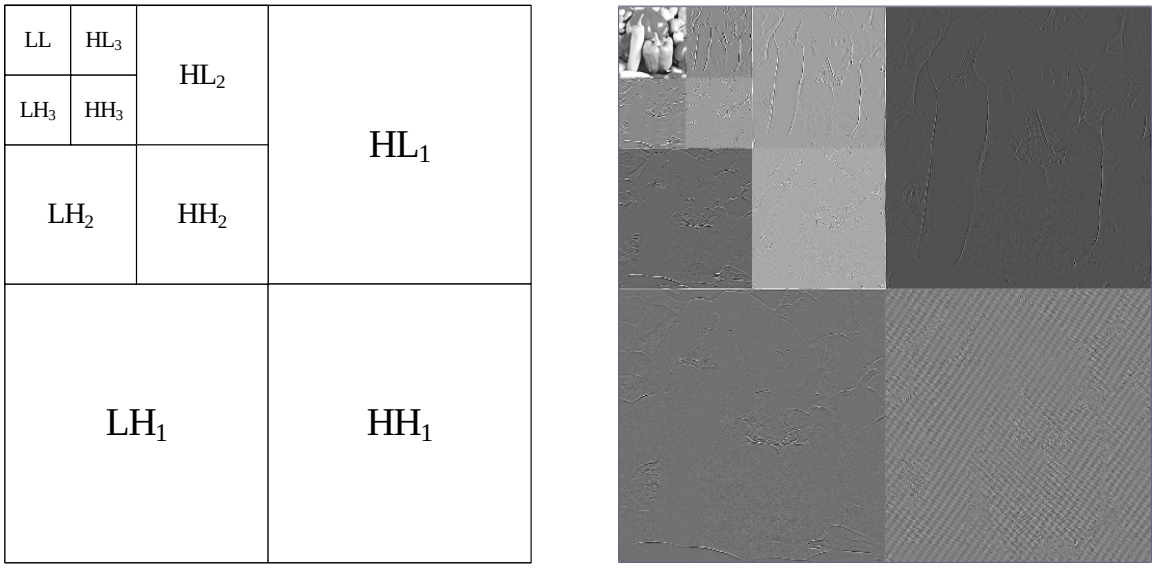


Figure 2.7 The block diagram of decomposition of two-dimensional wavelet transform with three decomposition levels



(a) (b)

Figure 2.8 (a) The subband structure of two-dimensional DWT with three levels and (b) the wavelet transformed version of the "Fruit" image with three decomposition levels using the Daubechies-4 kernel

2.3.2 Lifting implementation of the wavelet transform

The wavelet transform can be implemented by the lifting realization [49], [50]. The lifting implementation is widely used in image processing [44], [47], [48] and video technology [52], [53], [54], [55], [56]. It can save a large number of operations as compared with the filtering approach and guarantee perfect reconstruction. Figure 2.9 shows the realization diagram of lifting implementation for analysis and synthesis stages of the wavelet transform. During decomposition, it consists of three phases. The first phase is to split the input signal into odd and even samples. The second stage is that the even samples are used to predict the odd ones and the predicted samples are subtracted from the original odd samples to form the high frequency signal which is the prediction error. The final stage is the update process which is to add back the error signals to the even samples to form the low frequency signal. These can be considered as the averages of the input signal. The reconstruction performs the reverse procedure to find out the reconstructed signal.

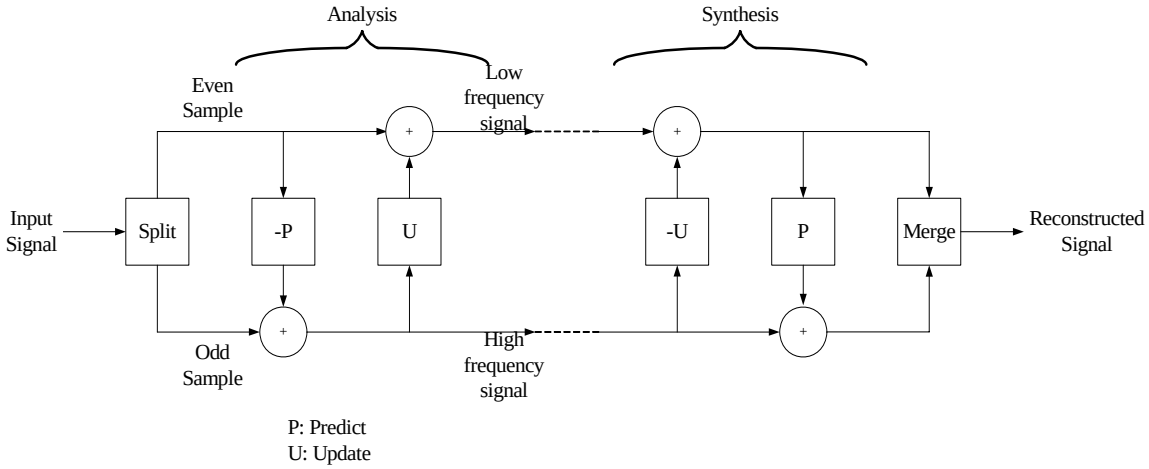


Figure 2.9 Block diagram of lifting implementation for decomposition and reconstruction of the wavelet transform

Figure 2.10 shows an example of lifting implementation of forward and backward wavelet transform for the Haar kernel. The forward lowpass and highpass filters can be written as,

$$L(z)=\frac{1}{2}+\frac{1}{2}z^{-1} \tag{2.17}$$

$$H(z)=\frac{1}{2}-\frac{1}{2}z^{-1} \tag{2.18}$$

where L and H are the lowpass and highpass filter respectively. The decomposition process of Haar kernel is,

$$\begin{aligned} split: & s_{j+1}^{0}[n]=s_j[2n] \\ & d_{j+1}^{0}[n]=s_j[2n+1] \\ predict: & d_{j+1}[n]=\frac{1}{2}\left(d_{j+1}^{0}[n]-s_{j+1}^{0}[n]\right) \\ update: & s_{j+1}[n]=s_{j+1}^{0}[n]+d_{j+1}[n] \end{aligned}$$

and the reconstruction operation of Haar kernel is,

$$\begin{aligned} update: & s_{j+1}^{0}[n]=s_{j+1}[n]-d_{j+1}[n] \\ predict: & d_{j+1}^{0}[n]=2d_{j+1}[n]+s_{j+1}^{0}[n] \\ merge: & s_j[2n]=s_{j+1}^{0}[n] \\ & s_j[2n+1]=d_{j+1}^{0}[n] \end{aligned}$$

where $s_j[2n]$ and $s_j[2n+1]$ are the even and odd samples respectively and $s_{j+1}[n]$ and $d_{j+1}[n]$ are the low and high frequency signals respectively.

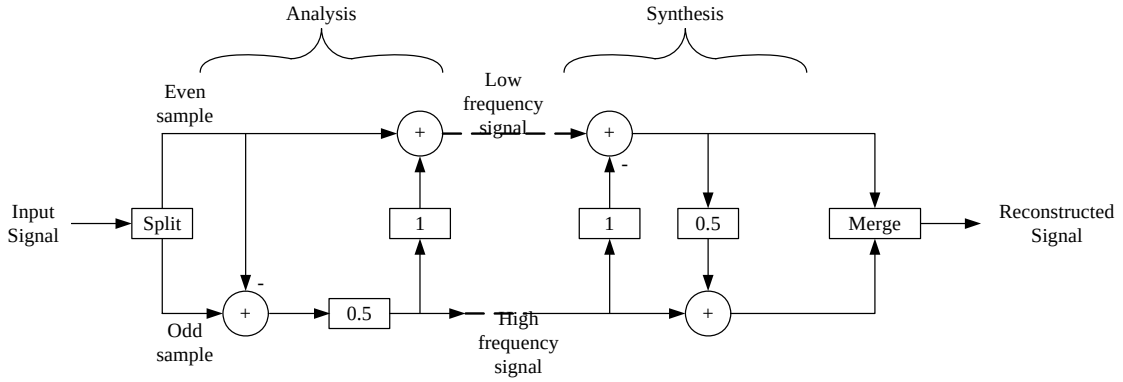


Figure 2.10 The realization diagram of lifting implement of the forward and backward wavelet transform for the Haar kernel

2.4 Literature review of the Embedded Zerotree Wavelet (EZW)

The motion estimation algorithms in the wavelet domain will be mentioned in chapters 3 and 4 and the Embedded Zerotree Wavelet (EZW) [57] is used in the wavelet video coder to get the experimental results. Besides, a modified approach of the EZW will be discussed in chapter 5. So, the algorithm of the EZW is reviewed in section 2.5.1 and an example is given in section 2.5.2.

Figure 2.11 (a) and (b) illustrate the block diagrams of the wavelet zerotree image encoder and decoder respectively. During encoding, the EZW [57] applies on the wavelet coefficients after carrying out wavelet decomposition. It converts the transformed coefficients into symbols and the definitions of the symbols will be explained in the following section. After that, the symbols will be entropy coded by arithmetic coding or Huffman coding. The decoder performs the reverse procedure to obtain the reconstructed image. The EZW can generate the encoded bit-stream in the order of significance. In order words, it can send the coefficients with large magnitude in prior of the coefficients with small magnitude to the decoder. It is embedded in nature and can achieve progressive transmission. Furthermore, the encoder can terminate the encoding procedures when the target bit rate is met. Also, the decoder can also stop the decoding procedures at any point of the truncated bitstream. Although the decoder cannot decode the intact bitstream, the quality of the reconstructed image is not considerably affected.

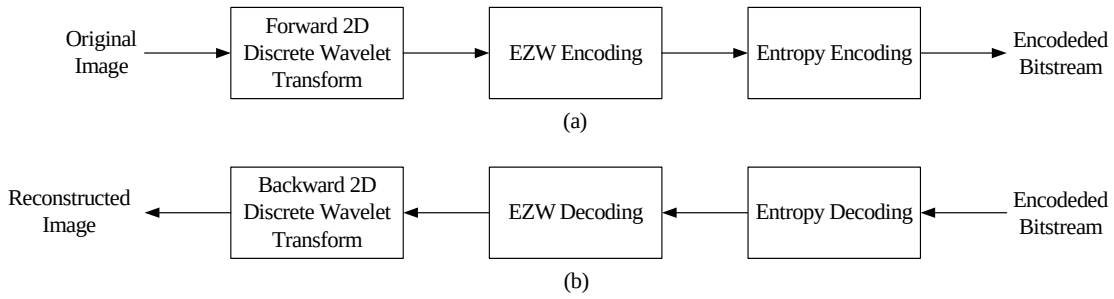


Figure 2.11 Block diagrams of the zerotree wavelet image (a) encoder and (b) decoder

EZW coding is commonly used in wavelet image or wavelet coder due to its superior compression performance. It is based on two major ideas which are the exploitation of the self-similarity inherent in the wavelet domain to predict the location of the significant information between different levels in the wavelet pyramid and successive approximation quantization (SAQ) of the wavelet coefficients. These two points refer to the dominant pass and subordinate pass respectively and they will be discussed into detail in the coming section.

2.4.1 EZW coding algorithm

The EZW algorithm re-allocates the wavelet coefficients by sending the significant coefficients before those coefficients with smaller magnitude. It uses the zerotree structure as depicted in Figure 2.12 to build up the parent and children relationship of the wavelet coefficients across different scales. In the wavelet pyramid, every coefficient at a certain level can be related to a set of coefficients of similar orientation at the next finer level. The zerotree can exploit the correlation of the wavelet coefficients among different levels in the wavelet pyramid. The coefficient at the coarse level is called the parent and all coefficients at the same spatial locations and of similar orientation at the next finer level are called children. For the lowest frequency subband,

LL subband, each parent node has three children, one in each subband at the same level and spatial location but a different orientation.

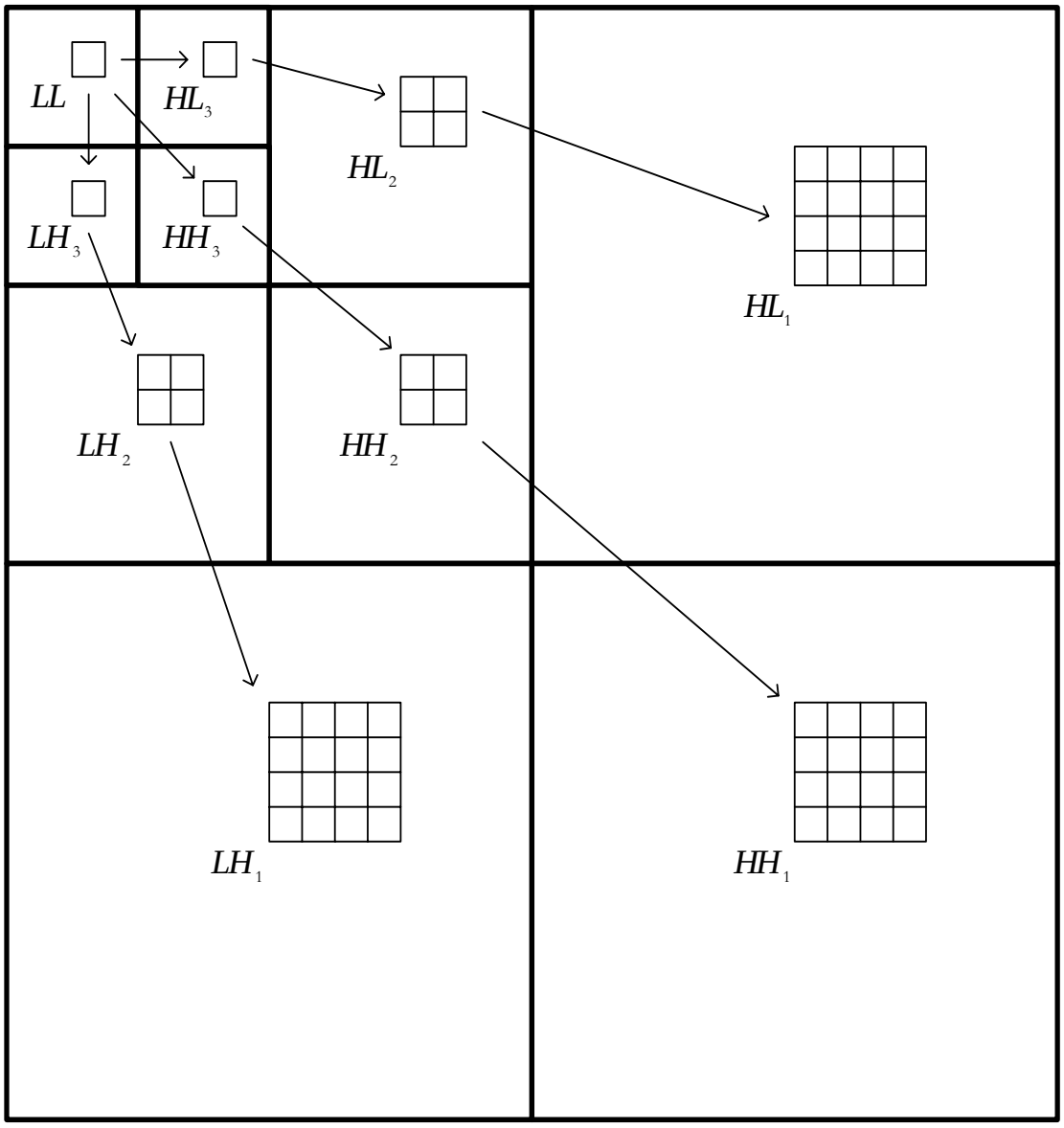


Figure 2.12 Parent and children relationship of the Zerotree

In order to achieve progressive transmission, the EZW coding progressively quantizes the wavelet coefficients by bit plane coding such that the most significant bit plane is transmitted to the decoder first. A bit plane is referred to a pass in the following. During the coding of EZW, the coefficients are scanned in zigzag order as shown in Figure 2.13. Furthermore, a threshold value, *T*, is used to determine the significant

coefficients in each pass. If the magnitude of coefficient is greater than the threshold, then it is defined as significant. Otherwise, it is categorized as insignificant. After processing a pass, the threshold is decreased by half. The iteration stops when the threshold value is equal to 0 or the target bit rate is achieved. The initial threshold, T_0, is usually set at $T_0 = 2^{\lfloor \log_2(\max|x_i|) \rfloor}$, where x_i is the largest coefficient of the transformed image in magnitude. Due to the progressive transmission, it consists of many passes. For each pass, it conveys the location of the significant and insignificant coefficients and the most significant bit of each significant coefficient by the Dominant Pass and Subordinate Pass respectively.

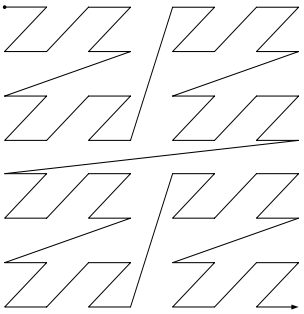


Figure 2.13 Zig-zag scanning of the wavelet coefficients

The Dominant Pass classifies the significant coefficients. Each coefficient is compared with the threshold value. If the coefficient is larger than the predefined threshold in magnitude, it will be identified as significant coefficient. Otherwise, it will be arranged as the insignificant coefficient. If the significant coefficient is larger than zero, the symbol “P” (Positive) will be assigned to that significant coefficient. If not, the symbol “N” (Negative) will be assigned to that coefficient. If the coefficient and all of its child nodes are smaller than the threshold value, the symbol “T” (Zerotree root) is consigned to that insignificant coefficient. If the coefficient is smaller than the threshold but one or more than one of its child nodes are larger than the threshold, the symbol “Z” (Isolated Zero) will be allocated to that coefficient. Once the coefficient is determined to

be significant, it is set to zero in order to prevent from coding again in the next pass. The magnitudes of the significant coefficients are put to the Subordinate List to perform Subordinate Pass. All child nodes of a zerotree root are not encoded. Therefore, the zerotree can encode the location of zero coefficients. Figure 2.14 illustrates the flow diagram of the Dominant Pass of the EZW coding.

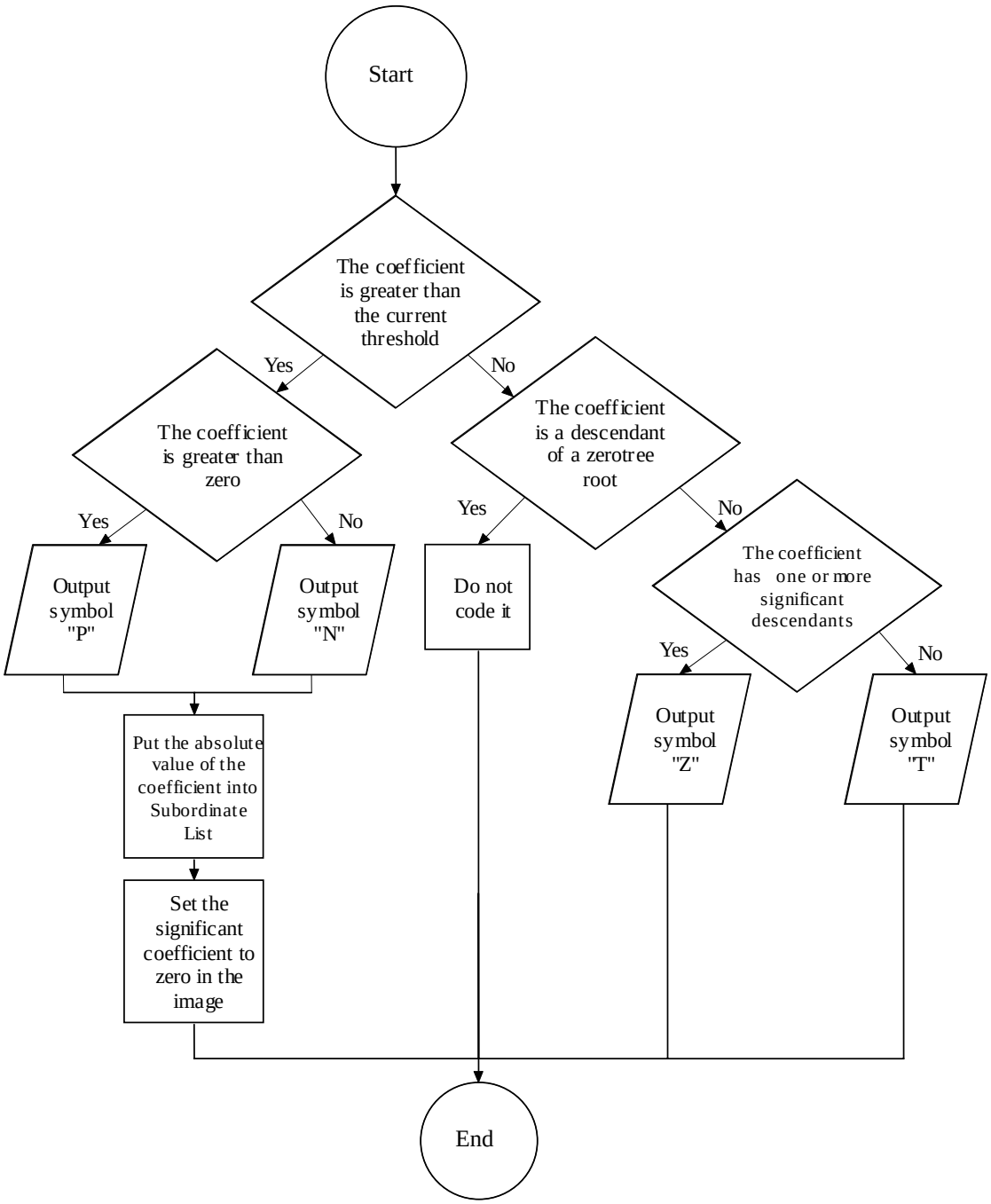


Figure 2.14 Flow chart of the dominant pass of the EZW encoding

The Subordinate Pass or the Refinement Pass provides one more bit on the magnitudes of the significant coefficients in the Subordinate List. The binary symbols “0” and “1” are given to the significant coefficients in the Subordinate List for the lower and upper half of the quantization intervals respectively. The limits of these quantization intervals are found by multiplying the current threshold by an integer k, where $k = 2, 3, 4, \ldots$ and these quantization intervals must be smaller than $2 \times T_0$. When the current threshold is equal to 1, the Subordinate Pass is skipped because it is not possible to categorize the significant coefficients into the lower or upper half of the quantization intervals. Figure 2.15 illustrates the flow chart of Subordinate Pass. An example will be mentioned below to explain the encoding and decoding procedures.

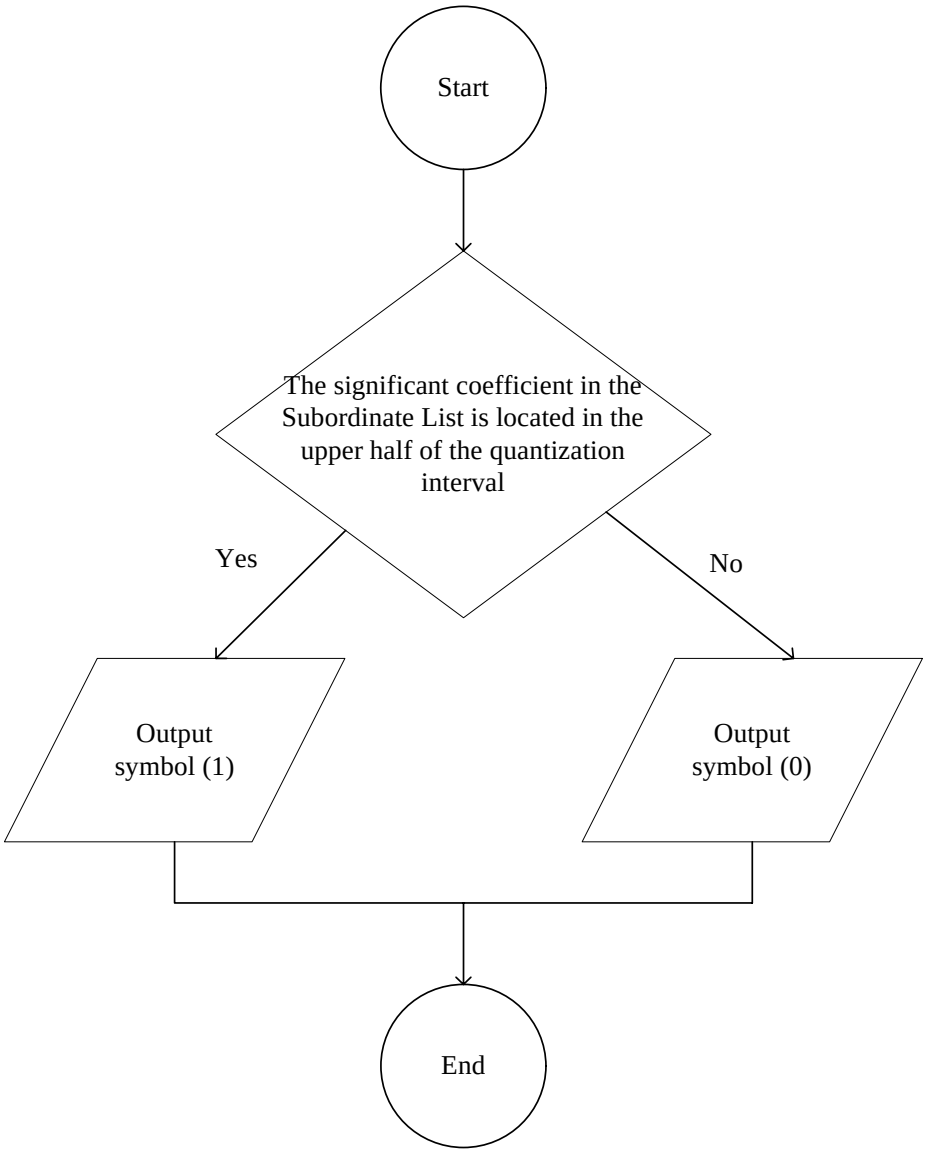


Figure 2.15 Flow chart of the subordinate pass of the EZW encoding

2.4.2 Encoding and decoding examples of EZW

A wavelet-transformed matrix will be shown as below which is an 8×8 matrix. The scanning pattern is in zigzag order. Three decomposition levels are performed.

118	22	7	20	-8	12	9	0
-12	18	-11	72	8	6	2	3
-59	11	19	9	7	-1	-2	8
12	-8	7	-8	-6	-4	-9	7
38	11	12	16	3	-2	2	1
-3	7	4	-5	4	2	-3	1
7	-2	0	7	2	5	0	2
8	8	-6	7	0	0	7	0

For the first pass, the initial threshold, T_0, is 64 calculated by the equation mentioned above. In the Dominant Pass, the first coefficient is 118 which is larger than T_0 and a symbol “P” is assigned to the Dominant List. The second coefficient is 22 which is smaller than T_0 but one of its descendants (72) is also greater than T_0 so that a symbol “Z” is assigned to the Dominant List. For the third coefficient (-12), itself and all of its descendants are smaller than T_0 so that a symbol “T” is assigned to the Dominant List to signify it as a zerotree node and all its descendants will not be encoded. After all coefficients are scanned in this way, the Dominant List of the first pass becomes **P Z T T T T T P T T T T**. When a coefficient is defined as significant, the absolute value of this significant coefficient is put into the Subordinate List to perform the Subordinate Pass. After the Dominant Pass, the Subordinate List becomes **(118, 72)**. Then, the coefficients put into the Subordinate List are carried out the Subordinate Pass in order to quantize the significant coefficients. The current threshold is 64, the Subordinate Threshold is 32 which is equal to the current threshold, 64, reduced by half and the interval used in the quantization is [64, 128]. When the significant coefficient is located in the upper half of the interval which is [64, 96], a “0” is allocated to the bitstream. When the significant coefficient is located in the lower half of the interval which is [96, 128], a “1” is assigned to the bitstream. Therefore, the bitstream after the first Subordinate Pass is **1 0**.The results after the first pass are

Pass 1:
Threshold = 64
Dominant List: P Z T T T T T P T T T T
Subordinate List: 1 0

For the second pass, the current threshold, T_1, is decreased by half which is 32. The significant coefficients defined in the previous pass are not scanned in this pass. Therefore, the matrix becomes

*	22	7	20	-8	12	9	0
-12	18	-11	*	8	6	2	3
-59	11	19	9	7	-1	-2	8
12	-8	7	-8	-6	-4	-9	7
38	11	12	16	3	-2	2	1
-3	7	4	-5	4	2	-3	1
7	-2	0	7	2	5	0	2
8	8	-6	7	0	0	7	0

In the Dominant Pass, it repeats the same procedures in the first pass. The first scanned coefficient (22) is smaller T_1 and all of its descendants are also smaller than T_1 so that a symbol "T" is assigned to the Dominant List. After the second Dominant Pass, the Dominant List is **T Z T N T T T P T T T**. The significant coefficients of the previous pass are also included in the Subordinate List. Thus, the Subordinate List becomes **(118, 72, 59, 38)**. In the Subordinate Pass, the current threshold is 32, the Subordinate Threshold is 16 and the quantization intervals are [32, 64), [64, 96) and [96, 128). The limits of these quantization intervals are found by multiplying the current threshold, T_1, by an integer k, where $k = 2, 3, 4, \ldots$ and these quantization intervals are smaller than $2 \times T_0$. For the first significant coefficient in the Subordinate List (118), it is located in the upper half [112, 128) of the quantizer [96, 128) so that a "1" is assigned to the bitstream. For the second significant coefficient (72), it is located in the lower half [64, 80) of the quantizer [64, 96) so that a "0" is assigned to the bitstream. The

remaining coefficients are repeated in this step to obtain the bitstream which is **1 0 1 0**. The results of the second pass are

Pass 2:
Threshold = 32
Dominant List: T Z T N T T T P T T T
Subordinate List: 1 0 1 0

The process is repeated until the current threshold is equal to 0 or the desired bitrate is obtained. The results of the remaining pass are

Pass 3:
Threshold = 16
Dominant List:
P Z P T P T Z T T P T T T T T T T T T T T T P T T T T T T
Subordinate List: 0 1 1 0 0 0 0 0 0

Pass 4:
Threshold = 8
Dominant List:
N Z N P P N P T N N P P T P T T T T T T T T P N T P T T P T T T T P P T T T T T T T T T T T T T T T T T T T T T
Subordinate List: 1 0 0 1 1 0 1 0 0 1 0 0 1 0 0 0 0 1 0 0 0 0 0 1 0 0

Pass 5:
Threshold = 4
Dominant List:
P P P T T T P T N N T P T P P N P T T P N P T T P T T T T T T P T T T T P T
Subordinate List:
1 0 1 1 1 1 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 1 1 1 0 1 1 0 0 1 1 1 1 0 0 1

Pass 6:
Threshold = 2
Dominant List:
T P P T N N N T P N P P T N T P T T T P T
Subordinate List:
0 0 1 0 0 0 0 1 0 0 1 1 0 0 1 0 0 0 0 1 0 1 1 0 0 0 1 1 0 1 0 0 1 1 0 1 1 1 0 1 0 1 1 0 1 0 1 0 1 0 0 0 1 0 0

Pass 7:
Threshold = 1
Dominant List: T N T P P T T T T
No subordinate pass!!

In the seventh pass, the Subordinate Threshold is equal to 0.5 which is not an integer so that the Subordinate Pass cannot not be performed in this pass. The encoder stops in this stage and the output bitstream is shown as **Header - DP1 - SP1 - DP2 - SP2 - DP3 - SP3 - DP4 - SP4 - DP5 - SP5 - DP6 - SP6 - DP7**, where Header is the initial threshold, T_0, DP1 and SP1 are the Dominant List and the bitstream of the Subordinate Pass in the first pass respectively.

At the decoder, it receives the header which contains the initial threshold, T_0, 64, then the DP1 and SP1. The quantization interval is [64, 128) in the encoder. It starts to reconstruct the image. In the first pass, the first symbol is “P” and the first bit is “1” so that the first reconstructed coefficient is 112 which is the mean value of the upper half of the quantizer [64, 128). The second symbol is “Z” so that the reconstructed coefficient is zero and all of its child nodes are scanned later because there is one or more than one of its child nodes are significant. The third symbol is “T” so that the reconstructed coefficient is zero and all of its descendants are not scanned and marked as “x” in the following matrix. The sixth symbol is also “P” and the second subordinate bit is “0” so that the reconstructed coefficient is 80 which is the mean value of the lower half of the quantizer [64, 128). After the first pass of decoding, the reconstructed matrix becomes

Pass 1:

112	0	0	0	x	x	x	x
0	0	0	80	x	x	x	x
x	x	x	x	x	x	0	0
x	x	x	x	x	x	0	0
x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x

In the second pass, the current threshold, T_1, is cut by half which is 32 and the decoder receives the DP2 and SP2 to reconstruct the matrix again. The decoder refines firstly the significant coefficients defined in the previous pass. The quantization intervals at this pass in the encoder are [32, 64), [64, 96) and [96, 128). As the first reconstructed coefficient in the previous pass is 112 which lies inside the interval [96, 128), so it refines to 120 which is the mean value of the upper half of the interval [96, 128) because the first subordinate bit in the bitstream is "1". For the second reconstructed coefficient in the previous pass (80), it refines to 72 which is the mean value of the lower half of the interval [64, 96) as 80 lies in the interval [64, 96) and the second bit of the bitstream is "0". After refining all significant coefficients defined in the previous pass, the remaining coefficients can be reconstructed by the DP2 and SP2. The first symbol inside DP2 is "T" so that the first reconstructed coefficient is zero and all of its descendants are not significant and not to be scanned. The fifth symbol is "N" and the third bit is "1" so that the fifth reconstructed coefficient is –56 because the decoder reconstructs the coefficients which is not decoded in the previous pass so that the interval should be $[T_1, 2\times T_1]$ which is [32, 64). After decoding all symbols in the DP2, the reconstructed matrix becomes

Pass 2:

120	0	x	x	x	x	x	x
0	0	x	72	x	x	x	x
-56	0	x	x	x	x	x	x
0	0	x	x	x	x	x	x
40	0	x	x	x	x	x	x
0	0	x	x	x	x	x	x
x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x

The decoder continues decoding the bitstream until the current threshold is zero or the desired bitrate is achieved. The results of decoding are shown as follows.

Pass 3:

116	20	0	20	x	x	0	0
0	20	0	76	x	x	0	0
-60	0	20	0	x	x	0	0
0	0	0	0	x	x	0	0
36	0	0	20	0	0	x	x
0	0	0	0	0	0	x	x
x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x

Pass 4:

118	22	0	22	-10	14	10	0
-14	18	-10	74	10	0	0	0
-58	10	18	10	0	0	0	10
14	-10	0	-10	0	0	-10	0
38	10	14	18	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	x	x	0	0
10	10	0	0	x	x	0	0

Pass 5:

119	23	7	21	-9	13	9	0
-13	19	-11	73	9	7	0	0
-59	11	19	9	7	0	0	9
13	-9	7	-9	-7	-5	-9	7
39	11	13	17	0	0	0	0
0	7	5	-5	5	0	0	0
7	0	0	7	0	5	0	0
9	9	-7	7	0	0	7	0

Pass 6:

118	22	7	20	-8	12	9	0
-12	18	-11	72	8	6	2	3
-59	11	19	9	7	0	-2	8
12	-8	7	-8	-6	-4	-9	7
38	11	12	16	3	-2	2	0
-3	7	4	-5	4	2	-3	0
7	-2	0	7	2	5	0	2
8	8	-6	7	0	0	7	0

Pass 7:

118	22	7	20	-8	12	9	0
-12	18	-11	72	8	6	2	3
-59	11	19	9	7	-1	-2	8
12	-8	7	-8	-6	-4	-9	7
38	11	12	16	3	-2	2	1
-3	7	4	-5	4	2	-3	1
7	-2	0	7	2	5	0	2
8	8	-6	7	0	0	7	0

The reconstructed matrix after decoding the encoded bitstream is exactly the same as the original matrix. Therefore, the EZW attains lossless coding if the encoder continues to encode until the current threshold is equal to one and the decoder refines the whole bitstream. Besides, it also realizes the progressive property since the most significant bit is transmitted to the decoder first.

2.5 Modifications of EZW

The conventional EZW algorithm was reviewed in the previous section. Three modified methods are discussed in this section. The objectives of them are to improve the compression efficiency and reduce the computational complexity.

2.5.1 Multi-threshold approach

For the conventional EZW algorithm, if the parent node is significant, i.e. greater than the threshold value in the current pass, then its child nodes will be scanned later. Similarly, when the parent node is insignificant, its child nodes will also be scanned in order to determine whether one or more than one of its child nodes are significant or not. If there are significant child nodes, then isolated zero (IZ) is assigned to that parent node and all of its child nodes will be scanned later no matter they are significant or insignificant. Otherwise, the zerotree root (ZTR) is assigned to that parent node. For example, if a parent node in LH subband at level two as depicted in Figure 2.16(b) is significant and all coefficients in the subband LH_1 are insignificant, then the subband LH_1 will still be scanned later in the conventional approach and the symbols generated in the subband LH_1 are encoded and conveyed to the decoder for reconstruction. Since all coefficients in the subband LH_1 are insignificant, so it needs not to be scanned and encoded.

According to this observation [58], a multi-threshold approach can be used to reduce the redundancy in the high frequency subbands leading to increase the compression efficiency. The multi-threshold is a set of coefficients with a maximum magnitude in each subband. As shown in Figure 2.16, the multi-threshold is {10, 15, … , 50, 70}. The coefficient with the maximum magnitude, i.e. the threshold, in each

subband can indicate whether all coefficients in that subband are insignificant or not. If that maximum coefficient is smaller than the current threshold, then all coefficients in that subband can be skipped without scanning again and this subband is defined as insignificant subband such as the HL_2 subband in Figure 2.16(b). Otherwise, it is classified as the significant subband, i.e. the LL subband in Figure 2.16(b). For the significant subband which is adjacent to the insignificant subband, it is determined as the marginal subband such as the HL_3 subband in Figure 2.16(b). Table 2.1(a) shows the codeword assignment to the symbols generated by the EZW coding for the significant subband. For the insignificant parent node in the significant subband, one more bit is used to determine whether its child nodes are significant or not. However, only one bit is enough to represent the insignificant parent node in the marginal subband as illustrated in Table 2.1(b) since all of its child nodes must be insignificant as indicated by the multi-threshold. Given that the redundancy in the high frequency subband is reduced, so the number of bits used to encode the coefficients in the high frequency subband can be saved. In this approach, the value of the multi-threshold is also transmitted to the decoder as side information for reconstruction and it is regarded as overhead.

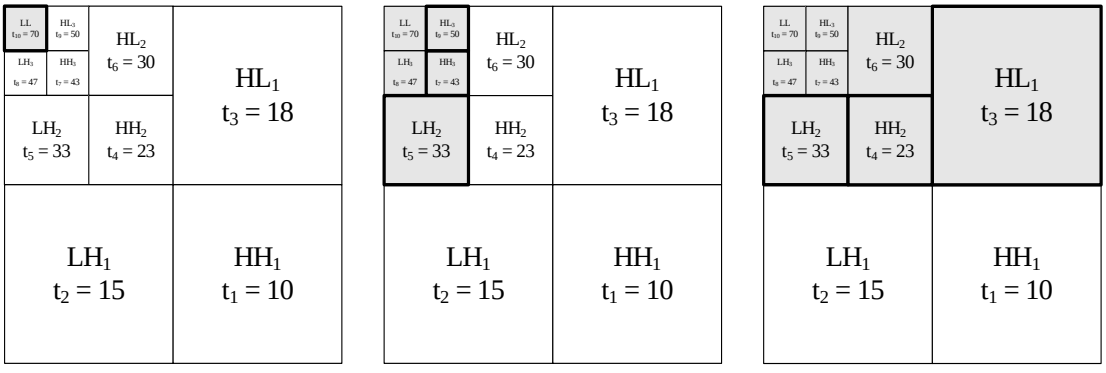


Figure 2.16 An example of the multi-threshold and significant subbands, where the shaded subbands and the subbands with think lines represent the insignificant subbands and marginal subbands respectively

Table 2.1 Codewords of the symbols generated by the multi-threshold EZW algorithm for (a) significant subbands and (b) marginal subbands

Symbols	Codewords
POS	11
NEG	10
IZ	01
ZTR	00

(a)

Symbols	Codewords
POS	11
NEG	10
ZTR	0
IZ	--

(b)

2.5.2 Fixed length residual value method

As mentioned in the section 2.4, the standard EZW algorithm uses four symbols to encode the wavelet coefficients. If a parent node is significant, then a symbol “P” or “N” is assigned to that coefficient depending on its sign and all of its child nodes will scanned later no matter they are insignificant or not. If all of its child nodes are insignificant, each coefficient will be given to a symbol (ZTR). Therefore, redundancy is introduced. The modified EZW algorithm with fixed length residual value [59] can be used to eliminate the redundant information.

For the modified scheme, eight symbols instead of four are used, which are described in Table 2.2. Figure 2.17 shows an example to illustrate the concept of the modified scheme. The first coefficient is 28 which is larger than the threshold, 16, and it is also greater than 1.5×threshold, 24, so a symbol “PP” is allocated to that coefficient. Besides, the residual value, 4 (28 – 24), is also encoded and transmitted to the decoder for reconstruction. For the second coefficient, 19, it is greater than the threshold and all of its child nodes are insignificant so that a symbol “PZR” is assigned to the second coefficient. Compared to the original EZW algorithm, only four symbols are used to encode the example in Figure 2.17 by using the modified scheme. The subordinate pass is eliminated in the modified algorithm. Instead, the residual value of the significant

coefficient is encoded and sent to the decoder. For each pass, the number of bits used to encode each residual value is $\log_2$ (threshold/2). In the example, it uses three bits to encode the residual value, 4, of the first coefficient, 28. Since the redundancy in the high frequency subband is removed, so the coding efficiency can be improved. Furthermore, the transmission of residual value can enhance the speed of encoding process.

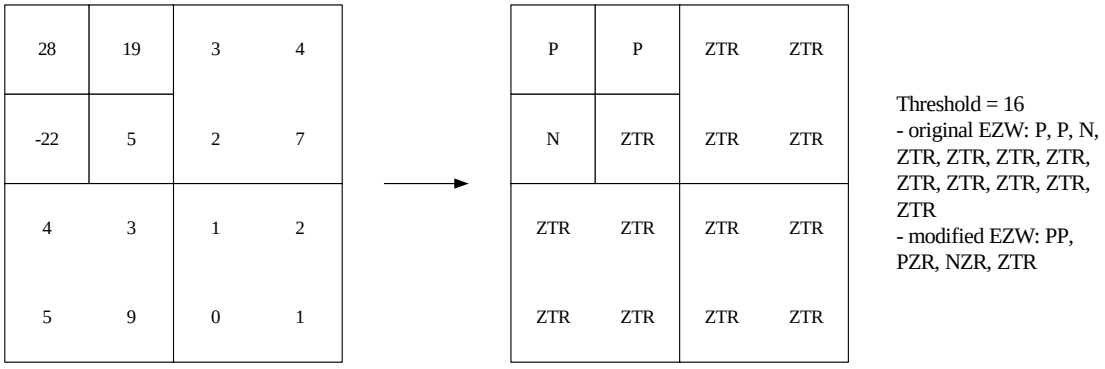


Figure 2.17 An example of the modified EZW algorithm using eight symbols in the dominant pass

Table 2.2 The descriptions of the eight symbols used in the modified EZW algorithm with fixed length residual value

Symbols	Description
IZ	The parent node is insignificant and one or more than one of its child nodes are significant
ZTR	The parent node and all of its child nodes are insignificant
PZR	The parent node is significant and positive valued and all of its child nodes are insignificant
NZR	The parent node is significant and negative valued and all of its child nodes are insignificant
PP	The parent node is significant, positive valued and greater than 1.5×threshold
PN	The parent node is significant, positive valued and smaller than 1.5×threshold
NP	The parent node is significant, negative valued and less than -1.5×threshold
NN	The parent node is significant, negative valued and the value of coefficient lies between –threshold and -1.5×threshold

2.5.3 Subband threshold scheme

The conventional EZW algorithm can losslessly encode the wavelet-transformed image when encoding all passes. The minimum weight subband method [60] can increase the coding gain with minimum loss during reconstruction. Figure 2.18(a) and

(b) depict the block diagrams of encoder and decoder using the minimum weight subband method respectively. The major difference to the conventional approach is that there is a pre-processing stage in the encoder. During pre-processing, some less important coefficients in the wavelet-transformed image are removed and these discarded coefficients cannot be recovered during reconstruction. The human visual system (HVS) cannot be aware of the degradation of the reconstructed image. However, these eliminated coefficients require more bits during compression. This method reduces some less important coefficients to attain further compression with slight quality degradation in the reconstructed image.

Usually, the HVS is more sensitive to the low frequency components than the high frequency components. According to this fact, the coefficients with small magnitude in the high frequency subband are removed. In the pre-processing stage, the weight of each subband is calculated, where the weight is the sum of the magnitudes of all coefficients in a subband, and the subband with minimum weight in each level is selected. After that, if a coefficient inside this minimum weight subband is smaller than a pre-defined threshold, it is set to zero. If the threshold is large, more coefficients will be set to zero and the reconstruction quality will become worse but the coding gain will be improved and vice versa. Therefore, the threshold should not be set too large and it usually lies between two to five. Since important information of the high frequency components, which contributes to the edge information, can still be preserved, so the visual quality of the minimum weight subband method can be comparable to that of the conventional EZW algorithm but the compression efficiency is improved due to the elimination of the less important coefficients in the high frequency subband.

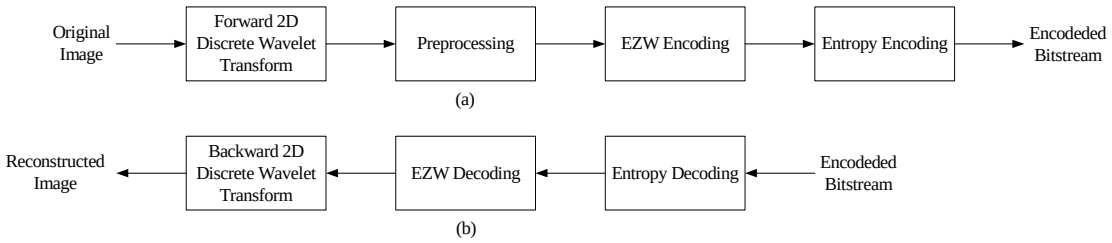


Figure 2.18 Block diagrams of the zerotree wavelet image (a) encoder and (b) decoder with minimum weight subband method

2.6 Literature review of Set-Partitioning Embedded Block Coder (SPECK)

One of the typical algorithms to encode the wavelet coefficients is the Embedded Zerotree Coding (EZW) algorithm which is mentioned in section 2.4. The EZW algorithm makes use of the parent and children relationships of subbands among different decomposition levels but the same orientation in the zerotree structure to achieve compression. The latest coding algorithm, Set-Partition Embedded Block Coder (SPECK) algorithm [107], does not use the tree structure which exploits the similarity across different subbands in the wavelet pyramid to encode the wavelet coefficients. The SPECK algorithm can achieve superior compression performance as compared with that of the EZW algorithm and preserve the wavelet coefficients in a lossless way. Also, it can attain progressive transmission such that the important wavelet coefficients are sent to the decoder first. Sections 2.6.1 mentions the coding methodology of SPECK algorithm and section 2.6.2 gives a numerical example to explain the encoding and decoding procedures of SPECK algorithm.

2.6.1 Coding methodology of SPECK algorithm

After applying Discrete Wavelet Transform (DWT) to an image *W*, the transformed image exhibits a pyramid structure defined by the levels of decomposition, the top most level of the wavelet pyramid becomes the root. The transformed coefficients, $\{c_{i,j}\}$, is located at (i, j) in the wavelet transformed image *X*. Like the EZW algorithm, the threshold value of first pass, T_0, is defined as, $T_0 = 2^n$, where $n = \lfloor \log_2 (\max |c_{i,j}|) \rfloor$. Actually, *n* is the total number of passes performed in the SPECK algorithm. A set *T* of coefficients is significant in the first pass if

$$\max_{(i,j)\in T} \{|c_{i,j}|\} \geq T_0$$

Otherwise, it is insignificant. Hence, the significance test of a set is defined as

$$\Gamma_n(T) = \begin{cases} 1, & \text{if } 2^n \leq \max_{(i,j)\in T} |c_{i,j}| < 2^{n+1} \\ 0, & \text{otherwise} \end{cases}$$

The SPECK algorithm contains two types of set which are type *S* and type *I*. These two sets come from the wavelet transformed image *X* as illustrated in Figure 2.19. A set *I* is decomposed into set *S* from coarser to finer resolution subbands through the transformed image. During the significance test, the EZW algorithm exploits the correlations among different subbands in the wavelet pyramid by making use of the zerotree structure. However, the SPECK algorithm does not use the tree structure. Instead, only the set *S* is performed the significance test. The objective of this approach is to utilize the clustering of energy found in the transformed image and concentrate on those areas of set which contain high energy. Thus, coefficients with high energy can be encoded first in order to achieve progressive transmission. Besides the sets *S* and *I*, the SPECK algorithm also keeps two lists which are the list of insignificant sets (LIS) and the list of significant pixels (LSP). LIS contains the sets *S* with different sizes which are

determined as insignificant to the threshold value in the current pass while LSP holds the significant coefficients which are defined to be significant in current or previous passes.

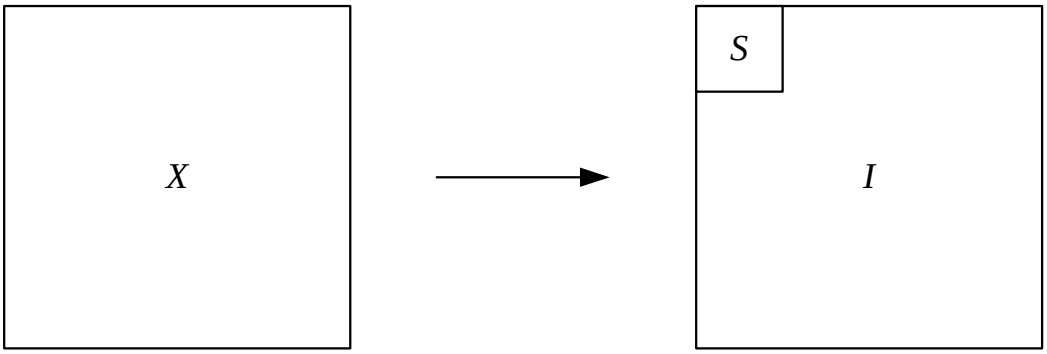


Figure 2.19 Partitioning of image X into sets S and I

After defining the symbols used in the SPECK algorithm, the main body of SPECK algorithm will be discussed. It consists of four major steps which are the initialization, sorting pass, refinement pass and quantization step. The first step is initialization which partitions the transformed image *X* into sets *S* and *I*. The set *S* represents the LL subband while the set *I* is the transformed image *X* excluding the set *S*. Then, the set *S* is added to the LIS and the LSP is set to empty set. Before going to sorting pass, the initial threshold, T_0, is calculated by this equation, $T_0 = 2^n$, where $n = \lfloor \log_2 (\max |c_{i,j}|) \rfloor$. In the sorting pass, each set *S* in the LIS is tested for significance. If a set *S* is significant with respect to the threshold value in current pass and is a coefficient, it is removed from the LIS and added to the LSP. A '1' together with the sign of that significant coefficient are output to the bitstream. If a set *S* is significant and is a set of coefficients, it is divided into four equal subsets, *O(S)*, as depicted in Figure 2.20. Each subset with size one-fourth of its parent set *S* is tested for significance again. This process is recursively performed until the significant coefficient is found. The insignificant sets *S* are put to the LIS and tested for significance with respect to the lower threshold value in the next passes and a '0' is allocated to the bitstream for each

insignificant set *S*. The aims to perform this process are that the coefficients with high energy can be determined and encoded first and the sets of insignificant coefficients can be classified such that the number of bits required to encode them can be limited. In other words, only a few number of bits are used to encode a large number of insignificant coefficients.

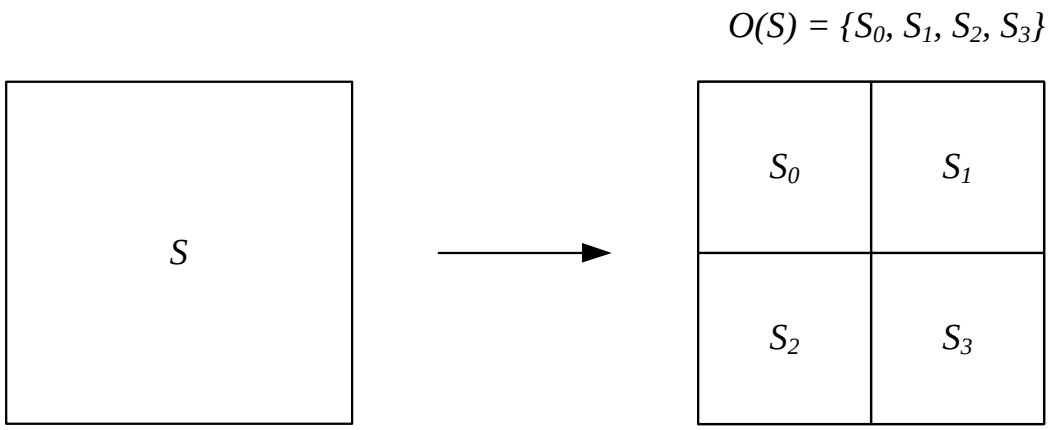


Figure 2.20 Partition of set S

After testing all sets *S*, the remainder set *I* is tested for significance against the threshold value in the current pass. If it is insignificant, then a '0' is assigned to the bitstream. Otherwise, it will be separated in four sets which are three new sets *S* and a new set *I* as shown in Figure 2.21 and a '1' is output to the bitstream. The size of each new set *S* is the same as the chopped part of transformed image *X*. Each new set *S* is tested for significance as described before. The new remainder set *I* is also tested for significance again until it is insignificant or it becomes an empty set. The philosophy of this scheme is to make use of the hierarchical structure in the wavelet pyramid which tells us that the energy is concentrated at the lower resolution subbands and decreases slowly when going down the wavelet pyramid. If a remainder set *I* is significant to the threshold value in the current pass, the significant coefficient is likely located in the top left region of *I*. Then, this region is split into new sets *S* which are tested for significance. As a result, the regions containing significant coefficients are grouped into

small sets and processed first while the regions have insignificant coefficients are formed into large sets which are only encoded by a few number of bits.

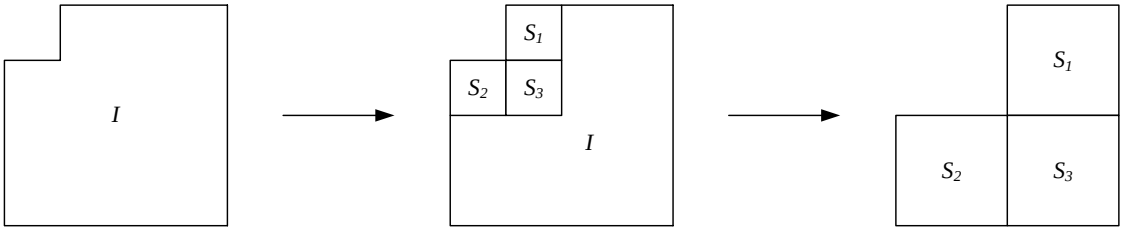


Figure 2.21 Partition of set I

The third step is refinement pass which is the same as the subordinate pass of EZW algorithm. Since section 2.4.1 discusses the subordinate pass of EZW algorithm in detail, so the refinement pass will not be mentioned in here. Finally, the threshold value of current pass is reduced by half and the LIS and LSP are passed to the next pass for further processing until the threshold value is equal to 1 or the desired bit rate is achieved. A detail example will be given in the coming section to explain the SPECK algorithm clearly.

2.6.2 Numerical example of encoding and decoding procedures of SPECK algorithm

Before discussing a coding example of SPECK algorithm, we define some symbols first. $S^k(i, j)$ and $(i, j)k$ denotes the point or set with the size of $2^k \times 2^k$ and with (i,j) upper left corner coordinate. (i, j) are the coordinates of a single coefficient and (i, j) in LSP is always a single point. We use an 8×8 matrix which is the same as that in section 2.4 to illustrate the operations of SPECK algorithm. The elements of matrix are the wavelet-transformed coefficients. The scanning pattern is in zigzag order. Three decomposition levels are performed.

118	22	7	20	-8	12	9	0
-12	18	-11	72	8	6	2	3
-59	11	19	9	7	-1	-2	8
12	-8	7	-8	-6	-4	-9	7
38	11	12	16	3	-2	2	1
-3	7	4	-5	4	2	-3	1
7	-2	0	7	2	5	0	2
8	8	-6	7	0	0	7	0

The maximum magnitude of coefficients is 118, so seven passes are carried out in total. For the first pass, the threshold value, T_0, is 64. The initial *S* set is the top left coefficient which is 118, so $S=S^0(0, 0)$ and (0, 0)0 initializes the List of Insignificant Set (LIS) and the List of Significant Pixels (LSP) is initially empty. The first coefficient (0, 0) is tested for significance. Its magnitude is 118, so it is significant. Hence, a '1' representing 'significant' and a '+' indicating its sign are sent to the bitstream and it is moved to the LSP. Then, the remainder set *I* is tested for significance, so a '1' is allocated to the bitstream and the remainder set *I* is partitioned into three new sets *S*, which are (0, 1), (1, 0) and (1, 1), and a new remainder set *I*. The three new sets *S* are insignificant, so a '0' is assigned to the bitstream for each insignificant set *S* and these three insignificant sets *S* are added to the LIS. Next, the remainder set *I* is tested for significance and it is significant. Thus, a '1' is sent to the bitstream and the remainder set *I* is divided into three new sets *S*, which are $S^1(0, 2)$, $S^1(2, 0)$ and $S^1(2, 2)$, and a new remainder set *I*. The set $S^1(0, 2)$ is tested for significance first. It is significant, so it is quadrisected into four singleton sets, which are (0, 2), (0, 3), (1, 2) and (1, 3), added to LIS and a '1' is output to the bitstream. Among these four coefficients, the first three coefficients, (0, 2), (0, 3) and (1, 2), are insignificant, so they are added to LIS and a '0' is assigned to each insignificant coefficient in the bitstream. Only the last one, (1, 3), is significant since its magnitude is 72, so it is put into the LSP and a '1+' is allocated to

the bitstream. The remaining two sets, $S^1(2, 0)$ and $S^1(2, 2)$, are insignificant, so they are appended in the LIS and a '0' is output to the bitstream for each insignificant set *S*. Finally, the remainder set *I* is insignificant, so a '0' is assigned in the bitstream. The following table shows the output bits of the sorting pass of first pass. Like the EZW algorithm, the significant coefficients are put to refinement pass and the refinement bits are **1 0**.

Comment	Point or Set	Output Bits	Action	Control Lists
S = (0, 0) I = rest				LIS = {(0, 0)0} LSP = Ø
	(0, 0)	1+	(0, 0) to LSP	LIS = Ø LSP = {(0, 0)}
Test I	S(I)	1	Split to 3'S, new I	
	(0, 1)	0	Add to LIS(0)	LIS = {(0, 1)0}
	(1, 0)	0	Add to LIS(0)	LIS = {(0, 1)0, (1, 0)0}
	(1, 1)	0	Add to LIS(0)	LIS = {(0, 1)0, (1, 0)0, (1, 1)0}
Test I	S(I)	1	Split to 3'S, new I	
	$S^1(0, 2)$	1	Quad split, add to LIS(0)	LIS = {(0, 1)0, (1, 0)0, (1, 1)0, (0, 2)0, (0, 3)0, (1, 2)0, (1, 3)0}
	(0, 2)	0		
	(0, 3)	0		
	(1, 2)	0		
	(1, 3)	1+	(1, 3) to LSP	LIS = {(0, 1)0, (1, 0)0, (1, 1)0, (0, 2)0, (0, 3)0, (1, 2)0} LSP = {(0, 0), (1, 3)}
	$S^1(2, 0)$	0	Add to LIS(1)	LIS = {(0, 1)0, (1, 0)0, (1, 1)0, (0, 2)0, (0, 3)0, (1, 2)0, (2, 0)1}
	$S^1(2, 2)$	0	Add to LIS(1)	LIS = {(0, 1)0, (1, 0)0, (1, 1)0, (0, 2)0, (0, 3)0, (1, 2)0, (2, 0)1, (2, 2)1}
Test I	S(I)	0		
Pass 1				LIS = {(0, 1)0, (1, 0)0, (1, 1)0, (0, 2)0, (0, 3)0, (1, 2)0, (2, 0)1, (2, 2)1} LSP = {(0, 0), (1, 3)}

The LIS at the end of the first pass is the initial list for second pass. The threshold value is this pass is halved, i.e. 32. The first six coefficients in the LIS, which are (0, 1), (1, 0), (1, 1), (0, 2), (0, 3) and (1, 2) are insignificant to the threshold value in this pass. So, a '0' is put to the bitstream for each insignificant coefficient. Then, the set $S^1(2, 0)$ is tested for significance and a '1' is assigned to the bitstream. Since it is

significant, so it is separated into four coefficients, (2, 0), (2, 1), (3, 0) and (3, 1), which are put to the LIS. The first coefficient, (2, 0), is greater than the threshold value in the current pass and is negative. Hence, it is removed in the LIS, put to the LSP and a '1-' is assigned to the bitstream. The last set in the LIS, $S^1(2, 2)$, is tested for significance. It is insignificant and a '0' is put to the bitstream. Next, the remainder set *I* is tested for significance and it is significant. Thus, it is broken up into three new sets *S*, $S^2(0, 4)$, $S^2(4, 0)$, $S^2(4, 4)$, which are added to the LIS, and a '1' is set to the bitstream. The first set, $S^2(0, 4)$, is tested and is insignificant. The second set, $S^2(4, 0)$, is significant, so it is further partitioned into four new sets, $S^1(4, 0)$, $S^1(4, 2)$, $S^1(6, 0)$ and $S^1(6, 2)$, which are put to the LIS and a '1' becomes the output bit. For the set $S^1(4, 0)$, it is significant and split into four coefficients, (4, 0), (4, 1), (5, 0) and (5, 1), which are put to the LIS. The first coefficient, (4, 0), is significant and positive, so it is added to the LSP and a '1+' is allocated to the bitstream. For the other three coefficients, (4, 1), (5, 0) and (5, 1), are insignificant. Thus, a '0' is appended to the bitstream for each insignificant coefficient. Next, the three sets, $S^1(4, 2)$, $S^1(6, 0)$ and $S^1(6, 2)$, are insignificant. Hence, a '0' is assigned to the bitstream for each insignificant set. Finally, the set, $S^2(4, 4)$ is also insignificant. Therefore, a '0' is also added to the bitstream. The output bits for the sorting pass of pass 2 are shown in the following table. Four coefficients are determined as significant coefficients and the refinement bits of refinement pass are **1 0 1 0**.

Comment	Point or Set	Output Bits	Action	Control Lists
				LIS = {(0, 1)0, (1, 0)0, (1, 1)0, (0, 2)0, (0, 3)0, (1, 2)0, (2, 0)1, (2, 2)1} LSP = {(0, 0), (1, 3)}
Test LIS(0)	(0, 1)	0		
	(1, 0)	0		
	(1, 1)	0		
	(0, 2)	0		
	(0, 3)	0		
	(1, 2)	0		
Test LIS(1)	S^1(2, 0)	1	Quad split, add to LIS(0)	LIS = {(0, 1)0, (1, 0)0, (1, 1)0, (0, 2)0, (0, 3)0, (1, 2)0, (2, 0)0, (2, 1)0, (3, 0)0, (3, 1)0, (2, 2)1}
	(2, 0)	1-	(2, 0) to LSP	LIS = {(0, 1)0, (1, 0)0, (1, 1)0, (0, 2)0, (0, 3)0, (1, 2)0, (2, 1)0, (3, 0)0, (3, 1)0, (2, 2)1} LSP = {(0, 0), (1, 3), (2, 0)}
	(2, 1)	0		
	(3, 0)	0		
	(3, 1)	0		
	S^1(2, 2)	0		
Test I	S(I)	1	Quad split, add to LIS(2)	LIS = {(0, 1)0, (1, 0)0, (1, 1)0, (0, 2)0, (0, 3)0, (1, 2)0, (2, 1)0, (3, 0)0, (3, 1)0, (2, 2)1, (0, 4)2, (4, 0)2, (4, 4)2}
	S^2(0, 4)	0		
	S^2(4, 0)	1	Quad split, add to LIS(1)	LIS = {(0, 1)0, (1, 0)0, (1, 1)0, (0, 2)0, (0, 3)0, (1, 2)0, (2, 1)0, (3, 0)0, (3, 1)0, (2, 2)1, (4, 0)1, (4, 2)1, (6, 0)1, (6, 2)1, (0, 4)2, (4, 4)2}
	S^1(4, 0)	1	Quad split, add to LIS(0)	LIS = {(0, 1)0, (1, 0)0, (1, 1)0, (0, 2)0, (0, 3)0, (1, 2)0, (2, 1)0, (3, 0)0, (3, 1)0, (4, 0)0, (4, 1)0, (5, 0)0, (5, 1)0, (2, 2)1, (4, 2)1, (6, 0)1, (6, 2)1, (0, 4)2, (4, 4)2}
	(4, 0)	1+	(4, 0) to LSP	LIS = {(0, 1)0, (1, 0)0, (1, 1)0, (0, 2)0, (0, 3)0, (1, 2)0, (2, 1)0, (3, 0)0, (3, 1)0, (4, 1)0, (5, 0)0, (5, 1)0, (2, 2)1, (4, 2)1, (6, 0)1, (6, 2)1, (0, 4)2, (4, 4)2} LSP = {(0, 0), (1, 3), (2, 0), (4, 0)}
	(4, 1)	0		
	(5, 0)	0		
	(5, 1)	0		
	S^1(4, 2)	0		
	S^1(6, 0)	0		
	S^1(6, 2)	0		
	S^2(4, 4)	0		
Pass 2				LIS = {(0, 1)0, (1, 0)0, (1, 1)0, (0, 2)0, (0, 3)0, (1, 2)0, (2, 1)0, (3, 0)0, (3, 1)0, (4, 1)0, (5, 0)0, (5, 1)0, (2, 2)1, (4, 2)1, (6, 0)1, (6, 2)1, (0, 4)2, (4, 4)2} LSP = {(0, 0), (1, 3), (2, 0), (4, 0)}

The process is repeated until the current threshold is equal to 1 or the desired bitrate is obtained. The results of the remaining passes are shown in the following tables.

Pass 3

Comment	Point or Set	Output Bits	Action	Control Lists
				LIS = {(0, 1)0, (1, 0)0, (1, 1)0, (0, 2)0, (0, 3)0, (1, 2)0, (2, 1)0, (3, 0)0, (3, 1)0, (4, 1)0, (5, 0)0, (5, 1)0, (2, 2)1, (4, 2)1, (6, 0)1, (6, 2)1, (0, 4)2, (4, 4)2} LSP = {(0, 0), (1, 3), (2, 0), (4, 0)}
Test LIS(0)	(0, 1)	1+	(0, 1) to LSP	LIS = {(1, 0)0, (1, 1)0, (0, 2)0, (0, 3)0, (1, 2)0, (2, 1)0, (3, 0)0, (3, 1)0, (4, 1)0, (5, 0)0, (5, 1)0, (2, 2)1, (4, 2)1, (6, 0)1, (6, 2)1, (0, 4)2, (4, 4)2} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1)}
	(1, 0)	0		
	(1, 1)	1+	(0, 1) to LSP	LIS = {(1, 0)0, (0, 2)0, (0, 3)0, (1, 2)0, (2, 1)0, (3, 0)0, (3, 1)0, (4, 1)0, (5, 0)0, (5, 1)0, (2, 2)1, (4, 2)1, (6, 0)1, (6, 2)1, (0, 4)2, (4, 4)2} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1)}
	(0, 2)	0		
	(0, 3)	1+	(0, 3) to LSP	LIS = {(1, 0)0, (0, 2)0, (1, 2)0, (2, 1)0, (3, 0)0, (3, 1)0, (4, 1)0, (5, 0)0, (5, 1)0, (2, 2)1, (4, 2)1, (6, 0)1, (6, 2)1, (0, 4)2, (4, 4)2} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3)}
	(1, 2)	0		
	(2, 1)	0		
	(3, 0)	0		
	(3, 1)	0		
	(4, 1)	0		
	(5, 0)	0		
	(5, 2)	0		
Test LIS(1)	$S^1(2, 2)$	1	Quad split, add to LIS(0)	LIS = {(1, 0)0, (0, 2)0, (1, 2)0, (2, 1)0, (3, 0)0, (3, 1)0, (4, 1)0, (5, 0)0, (5, 1)0, (2, 2)0, (2, 3)0, (3, 2)0, (3, 3)0, (4, 2)1, (6, 0)1, (6, 2)1, (0, 4)2, (4, 4)2} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3)}
	(2, 2)	1+	(2, 2) to LSP	LIS = {(1, 0)0, (0, 2)0, (1, 2)0, (2, 1)0, (3, 0)0, (3, 1)0, (4, 1)0, (5, 0)0, (5, 1)0, (2, 3)0, (3, 2)0, (3, 3)0, (4, 2)1, (6, 0)1, (6, 2)1, (0, 4)2, (4, 4)2} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2)}
	(2, 3)	0		
	(3, 2)	0		
	(3, 3)	0		
	$S^1(4, 2)$	1	Quad split, add to LIS(0)	LIS = {(1, 0)0, (0, 2)0, (1, 2)0, (2, 1)0, (3, 0)0, (3,

				1)0, (4, 1)0, (5, 0)0, (5, 1)0, (2, 3)0, (3, 2)0, (3, 3)0, (4, 2)0, (4, 3)0, (5, 2)0, (5, 3)0, (6, 0)1, (6, 2)1, (0, 4)2, (4, 4)2}
	(4, 2)	0		
	(4, 3)	1+	(4, 3) to LSP	LIS = {(1, 0)0, (0, 2)0, (1, 2)0, (2, 1)0, (3, 0)0, (3, 1)0, (4, 1)0, (5, 0)0, (5, 1)0, (2, 3)0, (3, 2)0, (3, 3)0, (4, 2)0, (5, 2)0, (5, 3)0, (6, 0)1, (6, 2)1, (0, 4)2, (4, 4)2} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3)}
	(5, 2)	0		
	(5, 3)	0		
	S^1(6, 0)	0		
	S^1(6, 2)	0		
Test LIS(2)	S^2(0, 4)	0		
	S^2(4, 4)	0		
Pass 3				LIS = {(1, 0)0, (0, 2)0, (1, 2)0, (2, 1)0, (3, 0)0, (3, 1)0, (4, 1)0, (5, 0)0, (5, 1)0, (2, 3)0, (3, 2)0, (3, 3)0, (4, 2)0, (5, 2)0, (5, 3)0, (6, 0)1, (6, 2)1, (0, 4)2, (4, 4)2} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3)}

Refinement Bits = 0 1 1 0 0 0 0 0 0

Pass 4

Comment	Point or Set	Output Bits	Action	Control Lists
				LIS = {(1, 0)0, (0, 2)0, (1, 2)0, (2, 1)0, (3, 0)0, (3, 1)0, (4, 1)0, (5, 0)0, (5, 1)0, (2, 3)0, (3, 2)0, (3, 3)0, (4, 2)0, (5, 2)0, (5, 3)0, (6, 0)1, (6, 2)1, (0, 4)2, (4, 4)2} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3)}
Test LIS(0)	(1, 0)	1-	(1, 0) to LSP	LIS = {(0, 2)0, (1, 2)0, (2, 1)0, (3, 0)0, (3, 1)0, (4, 1)0, (5, 0)0, (5, 1)0, (2, 3)0, (3, 2)0, (3, 3)0, (4, 2)0, (5, 2)0, (5, 3)0, (6, 0)1, (6, 2)1, (0, 4)2, (4, 4)2} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3), (1, 0)}
	(0, 2)	0		
	(1, 2)	1-	(1, 2) to LSP	LIS = {(0, 2)0, (2, 1)0, (3, 0)0, (3, 1)0, (4, 1)0, (5, 0)0, (5, 1)0, (2, 3)0, (3, 2)0, (3, 3)0, (4, 2)0, (5, 2)0, (5, 3)0, (6, 0)1, (6, 2)1, (0, 4)2, (4, 4)2} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3), (1, 0), (1, 2)}
	(2, 1)	1+	(2, 1) to LSP	LIS = {(0, 2)0, (3, 0)0, (3, 1)0, (4, 1)0, (5, 0)0, (5, 1)0, (2, 3)0, (3, 2)0, (3, 3)0, (4, 2)0, (5, 2)0, (5, 3)0, (6, 0)1, (6, 2)1 , (0, 4)2, (4, 4)2} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1),

				(0, 3), (2, 2), (4, 3), (1, 0), (1, 2), (2, 1)}
	(3, 0)	1+	(3, 0) to LSP	LIS = {(0, 2)0, (3, 1)0, (4, 1)0, (5, 0)0, (5, 1)0, (2, 3)0, (3, 2)0, (3, 3)0, (4, 2)0, (5, 2)0, (5, 3)0, (6, 0)1, (6, 2)1, (0, 4)2, (4, 4)2} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3), (1, 0), (1, 2), (2, 1), (3, 0)}
	(3, 1)	1-	(3, 1) to LSP	LIS = {(0, 2)0, (4, 1)0, (5, 0)0, (5, 1)0, (2, 3)0, (3, 2)0, (3, 3)0, (4, 2)0, (5, 2)0, (5, 3)0, (6, 0)1, (6, 2)1, (0, 4)2, (4, 4)2} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3), (1, 0), (1, 2), (2, 1), (3, 0), (3, 1)}
	(4, 1)	1+	(4, 1) to LSP	LIS = {(0, 2)0, (5, 0)0, (5, 1)0, (2, 3)0, (3, 2)0, (3, 3)0, (4, 2)0, (5, 2)0, (5, 3)0, (6, 0)1, (6, 2)1, (0, 4)2, (4, 4)2} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3), (1, 0), (1, 2), (2, 1), (3, 0), (3, 1), (4, 1)}
	(5, 0)	0		
	(5, 1)	0		
	(2, 3)	1+	(2, 3) to LSP	LIS = {(0, 2)0, (5, 0)0, (5, 1)0, (3, 2)0, (3, 3)0, (4, 2)0, (5, 2)0, (5, 3)0, (6, 0)1, (6, 2)1, (0, 4)2, (4, 4)2} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3), (1, 0), (1, 2), (2, 1), (3, 0), (3, 1), (4, 1), (2, 3)}
	(3, 2)	0		
	(3, 3)	1-	(3, 3) to LSP	LIS = {(0, 2)0, (5, 0)0, (5, 1)0, (3, 2)0, (4, 2)0, (5, 2)0, (5, 3)0, (6, 0)1, (6, 2)1, (0, 4)2, (4, 4)2} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3), (1, 0), (1, 2), (2, 1), (3, 0), (3, 1), (4, 1), (2, 3), (3, 3)}
	(4, 2)	1+	(4, 2) to LSP	LIS = {(0, 2)0, (5, 0)0, (5, 1)0, (3, 2)0, (5, 2)0, (5, 3)0, (6, 0)1, (6, 2)1, (0, 4)2, (4, 4)2} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3), (1, 0), (1, 2), (2, 1), (3, 0), (3, 1), (4, 1), (2, 3), (3, 3), (4, 2)}
	(5, 2)	0		
	(5, 3)	0		
Test LIS(1)	$S^1(6, 0)$	1	Quad split, add to LIS(0)	LIS = {(0, 2)0, (5, 0)0, (5, 1)0, (3, 2)0, (5, 2)0, (5, 3)0, (6, 0)0, (6, 1)0, (7, 0)0, (7, 1)0, (6, 2)1, (0, 4)2, (4, 4)2}
	(6, 0)	0		
	(6, 1)	0		
	(7, 0)	1+	(7, 0) to LSP	LIS = {(0, 2)0, (5, 0)0, (5, 1)0, (3, 2)0, (5, 2)0, (5, 3)0, (6, 0)0, (6, 1)0, (7, 1)0, (6, 2)1, (0, 4)2, (4, 4)2} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3), (1, 0), (1, 2), (2, 1), (3, 0), (3,

				1), (4, 1), (2, 3), (3, 3), (4, 2), (7, 0)}
	(7, 1)	1+	(7, 1) to LSP	LIS = {(0, 2)0, (5, 0)0, (5, 1)0, (3, 2)0, (5, 2)0, (5, 3)0, (6, 0)0, (6, 1)0, (6, 2)1, (0, 4)2, (4, 4)2} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3), (1, 0), (1, 2), (2, 1), (3, 0), (3, 1), (4, 1), (2, 3), (3, 3), (4, 2), (7, 0), (7, 1)}
	S^1(6, 2)	0		
Test LIS(2)	S^2(0, 4)	1	Quad split, add to LIS(1)	LIS = {(0, 2)0, (5, 0)0, (5, 1)0, (3, 2)0, (5, 2)0, (5, 3)0, (6, 0)0, (6, 1)0, (6, 2)1, (0, 4)1, (0, 6)1, (2, 4)1, (2, 6)1, (4, 4)2}
	S^1(0, 4)	1	Quad split, add to LIS(0)	LIS = {(0, 2)0, (5, 0)0, (5, 1)0, (3, 2)0, (5, 2)0, (5, 3)0, (6, 0)0, (6, 1)0, (0, 4)0, (0, 5)0, (1, 4)0, (1, 5)0, (6, 2)1, (0, 6)1, (2, 4)1, (2, 6)1, (4, 4)2}
	(0, 4)	1-	(0, 4) to LSP	LIS = {(0, 2)0, (5, 0)0, (5, 1)0, (3, 2)0, (5, 2)0, (5, 3)0, (6, 0)0, (6, 1)0, (0, 5)0, (1, 4)0, (1, 5)0, (6, 2)1, (0, 6)1, (2, 4)1, (2, 6)1, (4, 4)2} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3), (1, 0), (1, 2), (2, 1), (3, 0), (3, 1), (4, 1), (2, 3), (3, 3), (4, 2), (7, 0), (7, 1), (0, 4)}
	(0, 5)	1+	(0, 5) to LSP	LIS = {(0, 2)0, (5, 0)0, (5, 1)0, (3, 2)0, (5, 2)0, (5, 3)0, (6, 0)0, (6, 1)0, (1, 4)0, (1, 5)0, (6, 2)1, (0, 6)1, (2, 4)1, (2, 6)1, (4, 4)2} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3), (1, 0), (1, 2), (2, 1), (3, 0), (3, 1), (4, 1), (2, 3), (3, 3), (4, 2), (7, 0), (7, 1), (0, 4), (0, 5)}
	(1, 4)	1+	(1, 4) to LSP	LIS = {(0, 2)0, (5, 0)0, (5, 1)0, (3, 2)0, (5, 2)0, (5, 3)0, (6, 0)0, (6, 1)0, (1, 5)0, (6, 2)1, (0, 6)1, (2, 4)1, (2, 6)1, (4, 4)2} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3), (1, 0), (1, 2), (2, 1), (3, 0), (3, 1), (4, 1), (2, 3), (3, 3), (4, 2), (7, 0), (7, 1), (0, 4), (0, 5), (1, 4)}
	(1, 5)	0		
	S^1(0, 6)	1	Quad split, add to LIS(0)	LIS = {(0, 2)0, (5, 0)0, (5, 1)0, (3, 2)0, (5, 2)0, (5, 3)0, (6, 0)0, (6, 1)0, (1, 5)0, (0, 6)0, (0, 7)0, (1, 6)0, (1, 7)0, (6, 2)1, (2, 4)1, (2, 6)1, (4, 4)2}
	(0, 6)	1+	(0, 6) to LSP	LIS = {(0, 2)0, (5, 0)0, (5, 1)0, (3, 2)0, (5, 2)0, (5, 3)0, (6, 0)0, (6, 1)0, (1, 5)0, (0, 7)0, (1, 6)0, (1, 7)0, (6, 2)1, (2, 4)1, (2, 6)1, (4, 4)2} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3), (1, 0), (1, 2), (2, 1), (3, 0), (3, 1), (4, 1), (2, 3), (3, 3), (4, 2), (7, 0), (7, 1), (0, 4), (0, 5), (1, 4), (0, 6)}
	(0, 7)	0		
	(1, 6)	0		
	(1, 7)	0		
	S^1(2, 4)	0		
	S^1(2, 6)	0	Quad split, add to LIS(0)	LIS = {(0, 2)0, (5, 0)0, (5, 1)0, (3, 2)0, (5, 2)0, (5,

				3)0, (6, 0)0, (6, 1)0, (1, 5)0, (0, 7)0, (1, 6)0, (1, 7)0, (2, 6)0, (2, 7)0, (3, 6)0, (3, 7)0, (6, 2)1, (2, 4)1, (4, 4)2}
	(2, 6)	0		
	(2, 7)	1+		LIS = {(0, 2)0, (5, 0)0, (5, 1)0, (3, 2)0, (5, 2)0, (5, 3)0, (6, 0)0, (6, 1)0, (1, 5)0, (0, 7)0, (1, 6)0, (1, 7)0, (2, 6)0, (3, 6)0, (3, 7)0, (6, 2)1, (2, 4)1, (4, 4)2} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3), (1, 0), (1, 2), (2, 1), (3, 0), (3, 1), (4, 1), (2, 3), (3, 3), (4, 2), (7, 0), (7, 1), (0, 4), (0, 5), (1, 4), (0, 6), (2, 7)}
	(3, 6)	1-		LIS = {(0, 2)0, (5, 0)0, (5, 1)0, (3, 2)0, (5, 2)0, (5, 3)0, (6, 0)0, (6, 1)0, (1, 5)0, (0, 7)0, (1, 6)0, (1, 7)0, (2, 6)0, (3, 7)0, (6, 2)1, (2, 4)1, (4, 4)2} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3), (1, 0), (1, 2), (2, 1), (3, 0), (3, 1), (4, 1), (2, 3), (3, 3), (4, 2), (7, 0), (7, 1), (0, 4), (0, 5), (1, 4), (0, 6), (2, 7), (3, 6)}
	(3, 7)	0		
	$S^2(4, 4)$	0		
Pass 4				LIS = {(0, 2)0, (5, 0)0, (5, 1)0, (3, 2)0, (5, 2)0, (5, 3)0, (6, 0)0, (6, 1)0, (1, 5)0, (0, 7)0, (1, 6)0, (1, 7)0, (2, 6)0, (3, 7)0, (6, 2)1, (2, 4)1, (4, 4)2} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3), (1, 0), (1, 2), (2, 1), (3, 0), (3, 1), (4, 1), (2, 3), (3, 3), (4, 2), (7, 0), (7, 1), (0, 4), (0, 5), (1, 4), (0, 6), (2, 7), (3, 6)}

Refinement Bits = 1 0 0 1 1 0 1 0 0 1 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0

Pass 5

Comment	Point or Set	Output Bits	Action	Control Lists
				LIS = {(0, 2)0, (5, 0)0, (5, 1)0, (3, 2)0, (5, 2)0, (5, 3)0, (6, 0)0, (6, 1)0, (1, 5)0, (0, 7)0, (1, 6)0, (1, 7)0, (2, 6)0, (3, 7)0, (6, 2)1, (2, 4)1, (4, 4)2} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3), (1, 0), (1, 2), (2, 1), (3, 0), (3, 1), (4, 1), (2, 3), (3, 3), (4, 2), (7, 0), (7, 1), (0, 4), (0, 5), (1, 4), (0, 6), (2, 7), (3, 6)}
Test LIS(0)	(0, 2)	1+	(0, 2) to LSP	LIS = {(5, 0)0, (5, 1)0, (3, 2)0, (5, 2)0, (5, 3)0, (6, 0)0, (6, 1)0, (1, 5)0, (0, 7)0, (1, 6)0, (1, 7)0, (2, 6)0, (3, 7)0, (6, 2)1, (2, 4)1, (4, 4)2} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3), (1, 0), (1, 2), (2, 1), (3, 0), (3, 1), (4, 1), (2, 3), (3, 3), (4, 2), (7, 0), (7, 1), (0, 4), (0, 5), (1, 4), (0, 6), (2, 7), (3, 6), (0, 2)}
	(5, 0)	0		
	(5, 1)	1+	(5, 1) to LSP	LIS = {(5, 0)0, (3, 2)0, (5, 2)0, (5, 3)0, (6, 0)0, (6, 1)0, (1, 5)0, (0, 7)0, (1, 6)0, (1, 7)0, (2, 6)0, (3,

				7)0, (6, 2)1, (2, 4)1, (4, 4)2} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3), (1, 0), (1, 2), (2, 1), (3, 0), (3, 1), (4, 1), (2, 3), (3, 3), (4, 2), (7, 0), (7, 1), (0, 4), (0, 5), (1, 4), (0, 6), (2, 7), (3, 6), (0, 2), (5, 1)}
	(3, 2)	1+	(3, 2) to LSP	LIS = {(5, 0)0, (5, 2)0, (5, 3)0, (6, 0)0, (6, 1)0, (1, 5)0, (0, 7)0, (1, 6)0, (1, 7)0, (2, 6)0, (3, 7)0, (6, 2)1, (2, 4)1, (4, 4)2} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3), (1, 0), (1, 2), (2, 1), (3, 0), (3, 1), (4, 1), (2, 3), (3, 3), (4, 2), (7, 0), (7, 1), (0, 4), (0, 5), (1, 4), (0, 6), (2, 7), (3, 6), (0, 2), (5, 1), (3, 2)}
	(5, 2)	1+	(5, 2) to LSP	LIS = {(5, 0)0, (5, 3)0, (6, 0)0, (6, 1)0, (1, 5)0, (0, 7)0, (1, 6)0, (1, 7)0, (2, 6)0, (3, 7)0, (6, 2)1, (2, 4)1, (4, 4)2} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3), (1, 0), (1, 2), (2, 1), (3, 0), (3, 1), (4, 1), (2, 3), (3, 3), (4, 2), (7, 0), (7, 1), (0, 4), (0, 5), (1, 4), (0, 6), (2, 7), (3, 6), (0, 2), (5, 1), (3, 2), (5, 2)}
	(5, 3)	1-	(5, 3) to LSP	LIS = {(5, 0)0, (6, 0)0, (6, 1)0, (1, 5)0, (0, 7)0, (1, 6)0, (1, 7)0, (2, 6)0, (3, 7)0, (6, 2)1, (2, 4)1, (4, 4)2} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3), (1, 0), (1, 2), (2, 1), (3, 0), (3, 1), (4, 1), (2, 3), (3, 3), (4, 2), (7, 0), (7, 1), (0, 4), (0, 5), (1, 4), (0, 6), (2, 7), (3, 6), (0, 2), (5, 1), (3, 2), (5, 2), (5, 3)}
	(6, 0)	1+	(6, 0) to LSP	LIS = {(5, 0)0, (6, 1)0, (1, 5)0, (0, 7)0, (1, 6)0, (1, 7)0, (2, 6)0, (3, 7)0, (6, 2)1, (2, 4)1, (4, 4)2} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3), (1, 0), (1, 2), (2, 1), (3, 0), (3, 1), (4, 1), (2, 3), (3, 3), (4, 2), (7, 0), (7, 1), (0, 4), (0, 5), (1, 4), (0, 6), (2, 7), (3, 6), (0, 2), (5, 1), (3, 2), (5, 2), (5, 3), (6, 0)}
	(6, 1)	0		
	(1, 5)	1+	(1, 5) to LSP	LIS = {(5, 0)0, (6, 1)0, (0, 7)0, (1, 6)0, (1, 7)0, (2, 6)0, (3, 7)0, (6, 2)1, (2, 4)1, (4, 4)2} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3), (1, 0), (1, 2), (2, 1), (3, 0), (3, 1), (4, 1), (2, 3), (3, 3), (4, 2), (7, 0), (7, 1), (0, 4), (0, 5), (1, 4), (0, 6), (2, 7), (3, 6), (0, 2), (5, 1), (3, 2), (5, 2), (5, 3), (6, 0), (1, 5)}
	(0, 7)	0		
	(1, 6)	0		
	(1, 7)	0		
	(2, 6)	0		
	(3, 7)	1+	(3, 7) to LSP	LIS = {(5, 0)0, (6, 1)0, (0, 7)0, (1, 6)0, (1, 7)0, (2, 6)0, (6, 2)1, (2, 4)1, (4, 4)2}

				LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3), (1, 0), (1, 2), (2, 1), (3, 0), (3, 1), (4, 1), (2, 3), (3, 3), (4, 2), (7, 0), (7, 1), (0, 4), (0, 5), (1, 4), (0, 6), (2, 7), (3, 6), (0, 2), (5, 1), (3, 2), (5, 2), (5, 3), (6, 0), (1, 5), (3, 7)}
Test LIS(1)	S^1(6, 2)	1	Quad split, add to LIS(0)	LIS = {(5, 0)0, (6, 1)0, (0, 7)0, (1, 6)0, (1, 7)0, (2, 6)0, (6, 2)0, (6, 3)0, (7, 2)0, (7, 3)0, (2, 4)1, (4, 4)2}
	(6, 2)	0		
	(6, 3)	1+	(6, 3) to LSP	LIS = {(5, 0)0, (6, 1)0, (0, 7)0, (1, 6)0, (1, 7)0, (2, 6)0, (6, 2)0, (7, 2)0, (7, 3)0, (2, 4)1, (4, 4)2} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3), (1, 0), (1, 2), (2, 1), (3, 0), (3, 1), (4, 1), (2, 3), (3, 3), (4, 2), (7, 0), (7, 1), (0, 4), (0, 5), (1, 4), (0, 6), (2, 7), (3, 6), (0, 2), (5, 1), (3, 2), (5, 2), (5, 3), (6, 0), (1, 5), (3, 7), (6, 3)}
	(7, 2)	1-	(7, 2) to LSP	LIS = {(5, 0)0, (6, 1)0, (0, 7)0, (1, 6)0, (1, 7)0, (2, 6)0, (6, 2)0, (7, 3)0, (2, 4)1, (4, 4)2} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3), (1, 0), (1, 2), (2, 1), (3, 0), (3, 1), (4, 1), (2, 3), (3, 3), (4, 2), (7, 0), (7, 1), (0, 4), (0, 5), (1, 4), (0, 6), (2, 7), (3, 6), (0, 2), (5, 1), (3, 2), (5, 2), (5, 3), (6, 0), (1, 5), (3, 7), (6, 3), (7, 2)}
	(7, 3)	1+	(7, 3) to LSP	LIS = {(5, 0)0, (6, 1)0, (0, 7)0, (1, 6)0, (1, 7)0, (2, 6)0, (6, 2)0, (2, 4)1, (4, 4)2} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3), (1, 0), (1, 2), (2, 1), (3, 0), (3, 1), (4, 1), (2, 3), (3, 3), (4, 2), (7, 0), (7, 1), (0, 4), (0, 5), (1, 4), (0, 6), (2, 7), (3, 6), (0, 2), (5, 1), (3, 2), (5, 2), (5, 3), (6, 0), (1, 5), (3, 7), (6, 3), (7, 2), (7, 3)}
	S^1(2, 4)	1	Quad split, add to LIS(0)	LIS = {(5, 0)0, (6, 1)0, (0, 7)0, (1, 6)0, (1, 7)0, (2, 6)0, (6, 2)0, (2, 4)0, (2, 5)0, (3, 4)0, (3, 5)0, (4, 4)2}
	(2, 4)	1+	(2, 4) to LSP	LIS = {(5, 0)0, (6, 1)0, (0, 7)0, (1, 6)0, (1, 7)0, (2, 6)0, (6, 2)0, (2, 5)0, (3, 4)0, (3, 5)0, (4, 4)2} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3), (1, 0), (1, 2), (2, 1), (3, 0), (3, 1), (4, 1), (2, 3), (3, 3), (4, 2), (7, 0), (7, 1), (0, 4), (0, 5), (1, 4), (0, 6), (2, 7), (3, 6), (0, 2), (5, 1), (3, 2), (5, 2), (5, 3), (6, 0), (1, 5), (3, 7), (6, 3), (7, 2), (7, 3), (2, 4)}
	(2, 5)	0		
	(3, 4)	1-	(3, 4) to LSP	LIS = {(5, 0)0, (6, 1)0, (0, 7)0, (1, 6)0, (1, 7)0, (2, 6)0, (6, 2)0, (2, 5)0, (3, 5)0, (4, 4)2} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3), (1, 0), (1, 2), (2, 1), (3, 0), (3, 1), (4, 1), (2, 3), (3, 3), (4, 2), (7, 0), (7, 1), (0, 4), (0, 5), (1, 4), (0, 6), (2, 7), (3, 6), (0, 2), (5, 1), (3, 2), (5, 2), (5, 3), (6, 0), (1, 5), (3, 7), (6, 3), (7, 2),

				(7, 3), (2, 4), (3, 4)}
	(3, 5)	1-	(3, 5) to LSP	LIS = {(5, 0)0, (6, 1)0, (0, 7)0, (1, 6)0, (1, 7)0, (2, 6)0, (6, 2)0, (2, 5)0, (4, 4)2} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3), (1, 0), (1, 2), (2, 1), (3, 0), (3, 1), (4, 1), (2, 3), (3, 3), (4, 2), (7, 0), (7, 1), (0, 4), (0, 5), (1, 4), (0, 6), (2, 7), (3, 6), (0, 2), (5, 1), (3, 2), (5, 2), (5, 3), (6, 0), (1, 5), (3, 7), (6, 3), (7, 2), (7, 3), (2, 4), (3, 4), (3, 5)}
Test LIS(2)	$S^2(4, 4)$	1	Quad split, add to LIS(1)	LIS = {(5, 0)0, (6, 1)0, (0, 7)0, (1, 6)0, (1, 7)0, (2, 6)0, (6, 2)0, (2, 5)0, (4, 4)1, (4, 6)1, (6, 4)1, (6, 6)1}
	$S^1(4, 4)$	1	Quad split, add to LIS(0)	LIS = {(5, 0)0, (6, 1)0, (0, 7)0, (1, 6)0, (1, 7)0, (2, 6)0, (6, 2)0, (2, 5)0, (4, 4)0, (4, 5)0, (5, 4)0, (5, 5)0, (4, 6)1, (6, 4)1, (6, 6)1}
	(4, 4)	0		
	(4, 5)	0		
	(5, 4)	1+	(5, 4) to LSP	LIS = {(5, 0)0, (6, 1)0, (0, 7)0, (1, 6)0, (1, 7)0, (2, 6)0, (6, 2)0, (2, 5)0, (4, 4)0, (4, 5)0, (5, 5)0, (4, 6)1, (6, 4)1, (6, 6)1} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3), (1, 0), (1, 2), (2, 1), (3, 0), (3, 1), (4, 1), (2, 3), (3, 3), (4, 2), (7, 0), (7, 1), (0, 4), (0, 5), (1, 4), (0, 6), (2, 7), (3, 6), (0, 2), (5, 1), (3, 2), (5, 2), (5, 3), (6, 0), (1, 5), (3, 7), (6, 3), (7, 2), (7, 3), (2, 4), (3, 4), (3, 5), (5, 4)}
	(5, 5)	0		
	$S^1(4, 6)$	0		
	$S^1(6, 4)$	1	Quad split, add to LIS(0)	LIS = {(5, 0)0, (6, 1)0, (0, 7)0, (1, 6)0, (1, 7)0, (2, 6)0, (6, 2)0, (2, 5)0, (4, 4)0, (4, 5)0, (5, 5)0, (6, 4)0, (6, 5)0, (7, 4)0, (7, 5)0, (4, 6)1, (6, 6)1}
	(6, 4)	0		
	(6, 5)	1+	(6, 5) to LSP	LIS = {(5, 0)0, (6, 1)0, (0, 7)0, (1, 6)0, (1, 7)0, (2, 6)0, (6, 2)0, (2, 5)0, (4, 4)0, (4, 5)0, (5, 5)0, (6, 4)0, (7, 4)0, (7, 5)0, (4, 6)1, (6, 6)1} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3), (1, 0), (1, 2), (2, 1), (3, 0), (3, 1), (4, 1), (2, 3), (3, 3), (4, 2), (7, 0), (7, 1), (0, 4), (0, 5), (1, 4), (0, 6), (2, 7), (3, 6), (0, 2), (5, 1), (3, 2), (5, 2), (5, 3), (6, 0), (1, 5), (3, 7), (6, 3), (7, 2), (7, 3), (2, 4), (3, 4), (3, 5), (5, 4), (6, 5)}
	(7, 4)	0		
	(7, 5)	0		
	$S^1(6, 6)$	1	Quad split, add to LIS(0)	LIS = {(5, 0)0, (6, 1)0, (0, 7)0, (1, 6)0, (1, 7)0, (2, 6)0, (6, 2)0, (2, 5)0, (4, 4)0, (4, 5)0, (5, 5)0, (6, 4)0, (7, 4)0, (7, 5)0, (6, 6)0, (6, 7)0, (7, 6)0, (7, 7)0, (4, 6)1}
	(6, 6)	0		
	(6, 7)	0		

	(7, 6)	1+	(7, 6) to LSP	LIS = {(5, 0)0, (6, 1)0, (0, 7)0, (1, 6)0, (1, 7)0, (2, 6)0, (6, 2)0, (2, 5)0, (4, 4)0, (4, 5)0, (5, 5)0, (6, 4)0, (7, 4)0, (7, 5)0, (6, 6)0, (6, 7)0, (7, 7)0, (4, 6)1} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3), (1, 0), (1, 2), (2, 1), (3, 0), (3, 1), (4, 1), (2, 3), (3, 3), (4, 2), (7, 0), (7, 1), (0, 4), (0, 5), (1, 4), (0, 6), (2, 7), (3, 6), (0, 2), (5, 1), (3, 2), (5, 2), (5, 3), (6, 0), (1, 5), (3, 7), (6, 3), (7, 2), (7, 3), (2, 4), (3, 4), (3, 5), (5, 4), (6, 5), (7, 6)}
	(7, 7)	0		
Pass 5				LIS = {(5, 0)0, (6, 1)0, (0, 7)0, (1, 6)0, (1, 7)0, (2, 6)0, (6, 2)0, (2, 5)0, (4, 4)0, (4, 5)0, (5, 5)0, (6, 4)0, (7, 4)0, (7, 5)0, (6, 6)0, (6, 7)0, (7, 7)0, (4, 6)1} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3), (1, 0), (1, 2), (2, 1), (3, 0), (3, 1), (4, 1), (2, 3), (3, 3), (4, 2), (7, 0), (7, 1), (0, 4), (0, 5), (1, 4), (0, 6), (2, 7), (3, 6), (0, 2), (5, 1), (3, 2), (5, 2), (5, 3), (6, 0), (1, 5), (3, 7), (6, 3), (7, 2), (7, 3), (2, 4), (3, 4), (3, 5), (5, 4), (6, 5), (7, 6)}

Refinement Bits = 1 0 1 1 1 1 0 1 0 0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 1 1 1 1 1 1 1 1 0 0 0 1

Pass 6

Comment	Point or Set	Output Bits	Action	Control Lists
				LIS = {(5, 0)0, (6, 1)0, (0, 7)0, (1, 6)0, (1, 7)0, (2, 6)0, (6, 2)0, (2, 5)0, (4, 4)0, (4, 5)0, (5, 5)0, (6, 4)0, (7, 4)0, (7, 5)0, (6, 6)0, (6, 7)0, (7, 7)0, (4, 6)1} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3), (1, 0), (1, 2), (2, 1), (3, 0), (3, 1), (4, 1), (2, 3), (3, 3), (4, 2), (7, 0), (7, 1), (0, 4), (0, 5), (1, 4), (0, 6), (2, 7), (3, 6), (0, 2), (5, 1), (3, 2), (5, 2), (5, 3), (6, 0), (1, 5), (3, 7), (6, 3), (7, 2), (7, 3), (2, 4), (3, 4), (3, 5), (5, 4), (6, 5), (7, 6)}
Test LIS(0)	(5, 0)	1-	(5, 0) to LSP	LIS = {(6, 1)0, (0, 7)0, (1, 6)0, (1, 7)0, (2, 6)0, (6, 2)0, (2, 5)0, (4, 4)0, (4, 5)0, (5, 5)0, (6, 4)0, (7, 4)0, (7, 5)0, (6, 6)0, (6, 7)0, (7, 7)0, (4, 6)1} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3), (1, 0), (1, 2), (2, 1), (3, 0), (3, 1), (4, 1), (2, 3), (3, 3), (4, 2), (7, 0), (7, 1), (0, 4), (0, 5), (1, 4), (0, 6), (2, 7), (3, 6), (0, 2), (5, 1), (3, 2), (5, 2), (5, 3), (6, 0), (1, 5), (3, 7), (6, 3), (7, 2), (7, 3), (2, 4), (3, 4), (3, 5), (5, 4), (6, 5), (7, 6), (5, 0)}
	(6, 1)	1-	(6, 1) to LSP	LIS = {(0, 7)0, (1, 6)0, (1, 7)0, (2, 6)0, (6, 2)0, (2, 5)0, (4, 4)0, (4, 5)0, (5, 5)0, (6, 4)0, (7, 4)0, (7, 5)0, (6, 6)0, (6, 7)0, (7, 7)0, (4, 6)1}

				LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3), (1, 0), (1, 2), (2, 1), (3, 0), (3, 1), (4, 1), (2, 3), (3, 3), (4, 2), (7, 0), (7, 1), (0, 4), (0, 5), (1, 4), (0, 6), (2, 7), (3, 6), (0, 2), (5, 1), (3, 2), (5, 2), (5, 3), (6, 0), (1, 5), (3, 7), (6, 3), (7, 2), (7, 3), (2, 4), (3, 4), (3, 5), (5, 4), (6, 5), (7, 6), (5, 0), (6, 1)}
	(0, 7)	0		
	(1, 6)	1+	(1, 6) to LSP	LIS = {(0, 7)0, (1, 7)0, (2, 6)0, (6, 2)0, (2, 5)0, (4, 4)0, (4, 5)0, (5, 5)0, (6, 4)0, (7, 4)0, (7, 5)0, (6, 6)0, (6, 7)0, (7, 7)0, (4, 6)1} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3), (1, 0), (1, 2), (2, 1), (3, 0), (3, 1), (4, 1), (2, 3), (3, 3), (4, 2), (7, 0), (7, 1), (0, 4), (0, 5), (1, 4), (0, 6), (2, 7), (3, 6), (0, 2), (5, 1), (3, 2), (5, 2), (5, 3), (6, 0), (1, 5), (3, 7), (6, 3), (7, 2), (7, 3), (2, 4), (3, 4), (3, 5), (5, 4), (6, 5), (7, 6), (5, 0), (6, 1), (1, 6)}
	(1, 7)	1+	(1, 7) to LSP	LIS = {(0, 7)0, (2, 6)0, (6, 2)0, (2, 5)0, (4, 4)0, (4, 5)0, (5, 5)0, (6, 4)0, (7, 4)0, (7, 5)0, (6, 6)0, (6, 7)0, (7, 7)0, (4, 6)1} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3), (1, 0), (1, 2), (2, 1), (3, 0), (3, 1), (4, 1), (2, 3), (3, 3), (4, 2), (7, 0), (7, 1), (0, 4), (0, 5), (1, 4), (0, 6), (2, 7), (3, 6), (0, 2), (5, 1), (3, 2), (5, 2), (5, 3), (6, 0), (1, 5), (3, 7), (6, 3), (7, 2), (7, 3), (2, 4), (3, 4), (3, 5), (5, 4), (6, 5), (7, 6), (5, 0), (6, 1), (1, 6), (1, 7)}
	(2, 6)	1-	(2, 6) to LSP	LIS = {(0, 7)0, (6, 2)0, (2, 5)0, (4, 4)0, (4, 5)0, (5, 5)0, (6, 4)0, (7, 4)0, (7, 5)0, (6, 6)0, (6, 7)0, (7, 7)0, (4, 6)1} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3), (1, 0), (1, 2), (2, 1), (3, 0), (3, 1), (4, 1), (2, 3), (3, 3), (4, 2), (7, 0), (7, 1), (0, 4), (0, 5), (1, 4), (0, 6), (2, 7), (3, 6), (0, 2), (5, 1), (3, 2), (5, 2), (5, 3), (6, 0), (1, 5), (3, 7), (6, 3), (7, 2), (7, 3), (2, 4), (3, 4), (3, 5), (5, 4), (6, 5), (7, 6), (5, 0), (6, 1), (1, 6), (1, 7), (2, 6)}
	(6, 2)	0		
	(2, 5)	0		
	(4, 4)	1+	(4, 4) to LSP	LIS = {(0, 7)0, (6, 2)0, (2, 5)0, (4, 5)0, (5, 5)0, (6, 4)0, (7, 4)0, (7, 5)0, (6, 6)0, (6, 7)0, (7, 7)0, (4, 6)1} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3), (1, 0), (1, 2), (2, 1), (3, 0), (3, 1), (4, 1), (2, 3), (3, 3), (4, 2), (7, 0), (7, 1), (0, 4), (0, 5), (1, 4), (0, 6), (2, 7), (3, 6), (0, 2), (5, 1), (3, 2), (5, 2), (5, 3), (6, 0), (1, 5), (3, 7), (6, 3), (7, 2), (7, 3), (2, 4), (3, 4), (3, 5), (5, 4), (6, 5), (7, 6), (5, 0), (6, 1), (1, 6), (1, 7), (2, 6), (4, 4)}

	(4, 5)	1-	(4, 5) to LSP	LIS = {(0, 7)0, (6, 2)0, (2, 5)0, (5, 5)0, (6, 4)0, (7, 4)0, (7, 5)0, (6, 6)0, (6, 7)0, (7, 7)0, (4, 6)1} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3), (1, 0), (1, 2), (2, 1), (3, 0), (3, 1), (4, 1), (2, 3), (3, 3), (4, 2), (7, 0), (7, 1), (0, 4), (0, 5), (1, 4), (0, 6), (2, 7), (3, 6), (0, 2), (5, 1), (3, 2), (5, 2), (5, 3), (6, 0), (1, 5), (3, 7), (6, 3), (7, 2), (7, 3), (2, 4), (3, 4), (3, 5), (5, 4), (6, 5), (7, 6), (5, 0), (6, 1), (1, 6), (1, 7), (2, 6), (4, 4), (4, 5)}
	(5, 5)	1+	(5, 5) to LSP	LIS = {(0, 7)0, (6, 2)0, (2, 5)0, (6, 4)0, (7, 4)0, (7, 5)0, (6, 6)0, (6, 7)0, (7, 7)0, (4, 6)1} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3), (1, 0), (1, 2), (2, 1), (3, 0), (3, 1), (4, 1), (2, 3), (3, 3), (4, 2), (7, 0), (7, 1), (0, 4), (0, 5), (1, 4), (0, 6), (2, 7), (3, 6), (0, 2), (5, 1), (3, 2), (5, 2), (5, 3), (6, 0), (1, 5), (3, 7), (6, 3), (7, 2), (7, 3), (2, 4), (3, 4), (3, 5), (5, 4), (6, 5), (7, 6), (5, 0), (6, 1), (1, 6), (1, 7), (2, 6), (4, 4), (4, 5), (5, 5)}
	(6, 4)	1+	(6, 4) to LSP	LIS = {(0, 7)0, (6, 2)0, (2, 5)0, (7, 4)0, (7, 5)0, (6, 6)0, (6, 7)0, (7, 7)0, (4, 6)1} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3), (1, 0), (1, 2), (2, 1), (3, 0), (3, 1), (4, 1), (2, 3), (3, 3), (4, 2), (7, 0), (7, 1), (0, 4), (0, 5), (1, 4), (0, 6), (2, 7), (3, 6), (0, 2), (5, 1), (3, 2), (5, 2), (5, 3), (6, 0), (1, 5), (3, 7), (6, 3), (7, 2), (7, 3), (2, 4), (3, 4), (3, 5), (5, 4), (6, 5), (7, 6), (5, 0), (6, 1), (1, 6), (1, 7), (2, 6), (4, 4), (4, 5), (5, 5), (6, 4)}
	(7, 4)	0		
	(7, 5)	0		
	(6, 6)	0		
	(6, 7)	1+	(6, 7) to LSP	LIS = {(0, 7)0, (6, 2)0, (2, 5)0, (7, 4)0, (7, 5)0, (6, 6)0, (7, 7)0, (4, 6)1} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3), (1, 0), (1, 2), (2, 1), (3, 0), (3, 1), (4, 1), (2, 3), (3, 3), (4, 2), (7, 0), (7, 1), (0, 4), (0, 5), (1, 4), (0, 6), (2, 7), (3, 6), (0, 2), (5, 1), (3, 2), (5, 2), (5, 3), (6, 0), (1, 5), (3, 7), (6, 3), (7, 2), (7, 3), (2, 4), (3, 4), (3, 5), (5, 4), (6, 5), (7, 6), (5, 0), (6, 1), (1, 6), (1, 7), (2, 6), (4, 4), (4, 5), (5, 5), (6, 4), (6, 7)}
	(7, 7)	0		
Test LIS(1)	S^1(4, 6)	1	Quad split, add to LIS(0)	LIS = {(0, 7)0, (6, 2)0, (2, 5)0, (7, 4)0, (7, 5)0, (6, 6)0, (7, 7)0, (4, 6)0, (4, 7)0, (5, 6)0, (5, 7)0}
	(4, 6)	1+	(4, 6) to LSP	LIS = {(0, 7)0, (6, 2)0, (2, 5)0, (7, 4)0, (7, 5)0, (6, 6)0, (7, 7)0, (4, 7)0, (5, 6)0, (5, 7)0} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3), (1, 0), (1, 2), (2, 1), (3, 0), (3, 1), (4, 1), (2, 3), (3, 3), (4, 2), (7, 0), (7, 1), (0, 4), (0, 5), (1, 4), (0, 6), (2, 7), (3, 6), (0, 2), (5, 1), (3,

				2), (5, 2), (5, 3), (6, 0), (1, 5), (3, 7), (6, 3), (7, 2), (7, 3), (2, 4), (3, 4), (3, 5), (5, 4), (6, 5), (7, 6), (5, 0), (6, 1), (1, 6), (1, 7), (2, 6), (4, 4), (4, 5), (5, 5), (6, 4), (6, 7), (4, 6)}
	(4, 7)	0		
	(5, 6)	1-	(5, 6) to LSP	LIS = {(0, 7)0, (6, 2)0, (2, 5)0, (7, 4)0, (7, 5)0, (6, 6)0, (7, 7)0, (4, 7)0, (5, 7)0} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3), (1, 0), (1, 2), (2, 1), (3, 0), (3, 1), (4, 1), (2, 3), (3, 3), (4, 2), (7, 0), (7, 1), (0, 4), (0, 5), (1, 4), (0, 6), (2, 7), (3, 6), (0, 2), (5, 1), (3, 2), (5, 2), (5, 3), (6, 0), (1, 5), (3, 7), (6, 3), (7, 2), (7, 3), (2, 4), (3, 4), (3, 5), (5, 4), (6, 5), (7, 6), (5, 0), (6, 1), (1, 6), (1, 7), (2, 6), (4, 4), (4, 5), (5, 5), (6, 4), (6, 7), (4, 6), (5, 6)}
	(5, 7)	0		
Pass 6				LIS = {(0, 7)0, (6, 2)0, (2, 5)0, (7, 4)0, (7, 5)0, (6, 6)0, (7, 7)0, (4, 7)0, (5, 7)0} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3), (1, 0), (1, 2), (2, 1), (3, 0), (3, 1), (4, 1), (2, 3), (3, 3), (4, 2), (7, 0), (7, 1), (0, 4), (0, 5), (1, 4), (0, 6), (2, 7), (3, 6), (0, 2), (5, 1), (3, 2), (5, 2), (5, 3), (6, 0), (1, 5), (3, 7), (6, 3), (7, 2), (7, 3), (2, 4), (3, 4), (3, 5), (5, 4), (6, 5), (7, 6), (5, 0), (6, 1), (1, 6), (1, 7), (2, 6), (4, 4), (4, 5), (5, 5), (6, 4), (6, 7), (4, 6), (5, 6)}

Refinement Bits = 0 0 1 0 0 0 0 1 0 0 1 1 0 0 1 1 0 0 0 0 0 0 0 1 0 1 1 1 1 0 1 1 0 1 1 0 1 1 0 0 0 1 1 1 0 0 1 0 1 0 0 0 0 0 1

Pass 7

Comment	Point or Set	Output Bits	Action	Control Lists
				LIS = {(0, 7)0, (6, 2)0, (2, 5)0, (7, 4)0, (7, 5)0, (6, 6)0, (7, 7)0, (4, 7)0, (5, 7)0} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3), (1, 0), (1, 2), (2, 1), (3, 0), (3, 1), (4, 1), (2, 3), (3, 3), (4, 2), (7, 0), (7, 1), (0, 4), (0, 5), (1, 4), (0, 6), (2, 7), (3, 6), (0, 2), (5, 1), (3, 2), (5, 2), (5, 3), (6, 0), (1, 5), (3, 7), (6, 3), (7, 2), (7, 3), (2, 4), (3, 4), (3, 5), (5, 4), (6, 5), (7, 6), (5, 0), (6, 1), (1, 6), (1, 7), (2, 6), (4, 4), (4, 5), (5, 5), (6, 4), (6, 7), (4, 6), (5, 6)}
Test LIS(0)	(0, 7)	0		
	(6, 2)	0		
	(2, 5)	1-	(2, 5) to LSP	LIS = {(0, 7)0, (6, 2)0, (7, 4)0, (7, 5)0, (6, 6)0, (7, 7)0, (4, 7)0, (5, 7)0} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3), (1, 0), (1, 2), (2, 1), (3, 0), (3, 1), (4, 1), (2, 3), (3, 3), (4, 2), (7, 0), (7, 1), (0, 4), (0, 5), (1, 4), (0, 6), (2, 7), (3, 6), (0, 2), (5, 1), (3, 2), (5, 2), (5, 3), (6, 0), (1, 5), (3, 7), (6, 3), (7, 2), (7, 3), (2, 4), (3, 4), (3, 5), (5, 4), (6, 5), (7, 6), (5, 0), (6, 1), (1, 6), (1, 7), (2, 6), (4, 4), (4, 5), (5, 5), (6, 4), (6, 7), (4, 6), (5, 6), (2, 5)}
	(7, 4)	0		
	(7, 5)	0		
	(6, 6)	0		
	(7, 7)	0		
	(4, 7)	1+	(4, 7) to LSP	LIS = {(0, 7)0, (6, 2)0, (7, 4)0, (7, 5)0, (6, 6)0, (7, 7)0, (5, 7)0} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3), (1, 0), (1, 2), (2, 1), (3, 0), (3, 1), (4, 1), (2, 3), (3, 3), (4, 2), (7, 0), (7, 1), (0, 4), (0, 5), (1, 4), (0, 6), (2, 7), (3, 6), (0, 2), (5, 1), (3, 2), (5, 2), (5, 3), (6, 0), (1, 5), (3, 7), (6, 3), (7, 2), (7, 3), (2, 4), (3, 4), (3, 5), (5, 4), (6, 5), (7, 6), (5, 0), (6, 1), (1, 6), (1, 7), (2, 6), (4, 4), (4, 5), (5, 5), (6, 4), (6, 7), (4, 6), (5, 6), (2, 5), (4, 7)}
	(5, 7)	1+	(5, 7) to LSP	LIS = {(0, 7)0, (6, 2)0, (7, 4)0, (7, 5)0, (6, 6)0, (7, 7)0} LSP = {(0, 0), (1, 3), (2, 0), (4, 0), (0, 1), (1, 1), (0, 3), (2, 2), (4, 3), (1, 0), (1, 2), (2, 1), (3, 0), (3, 1), (4, 1), (2, 3), (3, 3), (4, 2), (7, 0), (7, 1), (0, 4), (0, 5), (1, 4), (0, 6), (2, 7), (3, 6), (0, 2), (5, 1), (3, 2), (5, 2), (5, 3), (6, 0), (1, 5), (3, 7), (6, 3), (7, 2), (7, 3), (2, 4), (3, 4), (3, 5), (5, 4), (6, 5), (7, 6), (5, 0), (6, 1), (1, 6), (1, 7), (2, 6), (4, 4), (4, 5), (5, 5), (6, 4), (6, 7), (4, 6), (5, 6), (2, 5), (4, 7), (5, 7)}

No refinement pass

In the seventh pass, the refinement pass cannot not be performed since the threshold value is one in this pass. The encoder stops in this pass and the output bitstream is shown as **Header - SB1 - RB1 - SB2 - RB2 - SB3 - RB3 - SB4 - RB4 - SB5 - RB5 - SB6 - RB6 - SB7**, where Header is the initial threshold, T_0, SB1 and RB1 are the output bits of sorting pass and refinement bits of refinement pass in the first pass respectively. The structure of bitstream is the same as that of EZW algorithm.

The decoder just duplicates the procedure of the encoder after received the encoded bitstream. If the word of the column "Output Bits" is replaced by "Input Bits" in the result tables of sorting pass in the encoder, the same table, as shown in the follows, can be constructed by the received bitstream. At the decoder, it receives the header which contains the initial threshold, T_0, 64, then the SB1 and RB1. The quantization interval is [64,128) in the decoder. The first input bit is '1+' and the first refinement bit is '1', so the first reconstructed coefficient, (0, 0), is a positive coefficient and lies in the upper half, $[96,128)$, of the quantization interval, $[64,128)$. Hence, the first reconstructed coefficient is 112 which is the mean value of the upper half, $[96,128)$, of the quantization interval, $[64,128)$. The second significant coefficient in this pass is located at (1, 3) and its input bits of sorting pass and refinement bit is '1+' and '0' respectively. Thus, it lies on the lower half, $[80,96)$, of the quantization interval, $[64,128)$. The reconstructed coefficient, (1, 3), is 80 which is the average value of the lower half, $[96,128)$, of the quantization interval, $[64,128)$. The remaining coefficients are reconstructed in the same way as the encoding process and the reconstructed matrix of the first pass is shown as follows.

Comment	Point or Set	Input Bits	Action	Control Lists
S = (0, 0) I = rest				LIS = {(0, 0)0} LSP = Ø
	(0, 0)	1+	(0, 0) to LSP	LIS = Ø LSP = {(0, 0)}
Test I	S(I)	1	Split to 3'S, new I	
	(0, 1)	0	Add to LIS(0)	LIS = {(0, 1)0}
	(1, 0)	0	Add to LIS(0)	LIS = {(0, 1)0, (1, 0)0}
	(1, 1)	0	Add to LIS(0)	LIS = {(0, 1)0, (1, 0)0, (1, 1)0}
Test I	S(I)	1	Split to 3'S, new I	
	S^1(0, 2)	1	Quad split, add to LIS(0)	LIS = {(0, 1)0, (1, 0)0, (1, 1)0, (0, 2)0, (0, 3)0, (1, 2)0, (1, 3)0}
	(0, 2)	0		
	(0, 3)	0		
	(1, 2)	0		
	(1, 3)	1+	(1, 3) to LSP	LIS = {(0, 1)0, (1, 0)0, (1, 1)0, (0, 2)0, (0, 3)0, (1, 2)0} LSP = {(0, 0), (1, 3)}
	S^1(2, 0)	0	Add to LIS(1)	LIS = {(0, 1)0, (1, 0)0, (1, 1)0, (0, 2)0, (0, 3)0, (1, 2)0, (2, 0)1}
	S^1(2, 2)	0	Add to LIS(1)	LIS = {(0, 1)0, (1, 0)0, (1, 1)0, (0, 2)0, (0, 3)0, (1, 2)0, (2, 0)1, (2, 2)1}
Test I	S(I)	0		
Pass 1				LIS = {(0, 1)0, (1, 0)0, (1, 1)0, (0, 2)0, (0, 3)0, (1, 2)0, (2, 0)1, (2, 2)1} LSP = {(0, 0), (1, 3)}

Refinement Bits = 1 0

Reconstructed Matrix of Pass 1

112	0	0	0	0	0	0	0
0	0	0	80	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

For the second pass, the threshold value is halved, i.e. 32, and the significant coefficients determined in the previous pass are refined first. The first significant coefficient is located at (0, 0). Its value is 112 and the first refinement bit in this pass is '1', so it lies in the upper half, [112,128), of the quantization interval, [96,128). Thus,

its reconstructed value in this pass is 120 which is the mean value of the interval, [112,128). The remaining coefficients are reconstructed in this way. The reconstructed matrix of pass 2 is shown as follows.

Pass 2

Comment	Point or Set	Input Bits	Action	Control Lists
				LIS = {(0, 1)0, (1, 0)0, (1, 1)0, (0, 2)0, (0, 3)0, (1, 2)0, (2, 0)1, (2, 2)1} LSP = {(0, 0), (1, 3)}
Test LIS(0)	(0, 1)	0		
	(1, 0)	0		
	(1, 1)	0		
	(0, 2)	0		
	(0, 3)	0		
	(1, 2)	0		
Test LIS(1)	S^1(2, 0)	1	Quad split, add to LIS(0)	LIS = {(0, 1)0, (1, 0)0, (1, 1)0, (0, 2)0, (0, 3)0, (1, 2)0, (2, 0)0, (2, 1)0, (3, 0)0, (3, 1)0, (2, 2)1}
	(2, 0)	1-	(2, 0) to LSP	LIS = {(0, 1)0, (1, 0)0, (1, 1)0, (0, 2)0, (0, 3)0, (1, 2)0, (2, 1)0, (3, 0)0, (3, 1)0, (2, 2)1} LSP = {(0, 0), (1, 3), (2, 0)}
	(2, 1)	0		
	(3, 0)	0		
	(3, 1)	0		
	S^1(2, 2)	0		
Test I	S(I)	1	Quad split, add to LIS(2)	LIS = {(0, 1)0, (1, 0)0, (1, 1)0, (0, 2)0, (0, 3)0, (1, 2)0, (2, 1)0, (3, 0)0, (3, 1)0, (2, 2)1, (0, 4)2, (4, 0)2, (4, 4)2}
	S^2(0, 4)	0		
	S^2(4, 0)	1	Quad split, add to LIS(1)	LIS = {(0, 1)0, (1, 0)0, (1, 1)0, (0, 2)0, (0, 3)0, (1, 2)0, (2, 1)0, (3, 0)0, (3, 1)0, (2, 2)1, (4, 0)1, (4, 2)1, (6, 0)1, (6, 2)1, (0, 4)2, (4, 4)2}
	S^1(4, 0)	1	Quad split, add to LIS(0)	LIS = {(0, 1)0, (1, 0)0, (1, 1)0, (0, 2)0, (0, 3)0, (1, 2)0, (2, 1)0, (3, 0)0, (3, 1)0, (4, 0)0, (4, 1)0, (5, 0)0, (5, 1)0, (2, 2)1, (4, 2)1, (6, 0)1, (6, 2)1, (0, 4)2, (4, 4)2}
	(4, 0)	1+	(4, 0) to LSP	LIS = {(0, 1)0, (1, 0)0, (1, 1)0, (0, 2)0, (0, 3)0, (1, 2)0, (2, 1)0, (3, 0)0, (3, 1)0, (4, 1)0, (5, 0)0, (5, 1)0, (2, 2)1, (4, 2)1, (6, 0)1, (6, 2)1, (0, 4)2, (4, 4)2} LSP = {(0, 0), (1, 3), (2, 0), (4, 0)}
	(4, 1)	0		
	(5, 0)	0		
	(5, 1)	0		
	S^1(4, 2)	0		

	$S^1(6, 0)$	0		
	$S^1(6, 2)$	0		
	$S^2(4, 4)$	0		
Pass 2				LIS = {(0, 1)0, (1, 0)0, (1, 1)0, (0, 2)0, (0, 3)0, (1, 2)0, (2, 1)0, (3, 0)0, (3, 1)0, (4, 1)0, (5, 0)0, (5, 1)0, (2, 2)1, (4, 2)1, (6, 0)1, (6, 2)1, (0, 4)2, (4, 4)2} LSP = {(0, 0), (1, 3), (2, 0), (4, 0)}

Refinement Bits = 1 0 1 0

Reconstructed Matrix of Pass 2

120	0	0	0	0	0	0	0
0	0	0	72	0	0	0	0
-56	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
40	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

The remaining passes perform the similar procedure to reconstruct the matrix and the reconstructed matrixes in each pass are shown as follows.

Reconstructed Matrix of Pass 3

116	20	0	20	0	0	0	0
0	20	0	76	0	0	0	0
-60	0	20	0	0	0	0	0
0	0	0	0	0	0	0	0
36	0	0	20	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Reconstructed Matrix of Pass 4

118	22	0	22	-10	14	10	0
-14	18	-10	74	10	0	0	0
-58	10	18	10	0	0	0	10
14	-10	0	-10	0	0	-10	0
38	10	14	18	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
10	10	0	0	0	0	0	0

Reconstructed Matrix of Pass 5

119	23	7	21	-9	13	9	0
-13	19	-11	73	9	7	0	0
-59	11	19	9	7	0	0	9
13	-9	7	-9	-7	-5	-9	7
39	11	13	17	0	0	0	0
0	7	5	-5	5	0	0	0
7	0	0	7	0	5	0	0
9	9	-7	7	0	0	7	0

Reconstructed Matrix of Pass 6

118	22	7	20	-8	12	9	0
-12	18	-11	72	8	6	2	3
-59	11	19	9	7	0	-2	8
12	-8	7	-8	-6	-4	-9	7
38	11	12	16	3	-2	2	0
-3	7	4	-5	4	2	-3	0
7	-2	0	7	2	5	0	2
8	8	-6	7	0	0	7	0

Reconstructed Matrix of Pass 7

118	22	7	20	-8	12	9	0
-12	18	-11	72	8	6	2	3
-59	11	19	9	7	-1	-2	8
12	-8	7	-8	-6	-4	-9	7
38	11	12	16	3	-2	2	1
-3	7	4	-5	4	2	-3	1
7	-2	0	7	2	5	0	2
8	8	-6	7	0	0	7	0

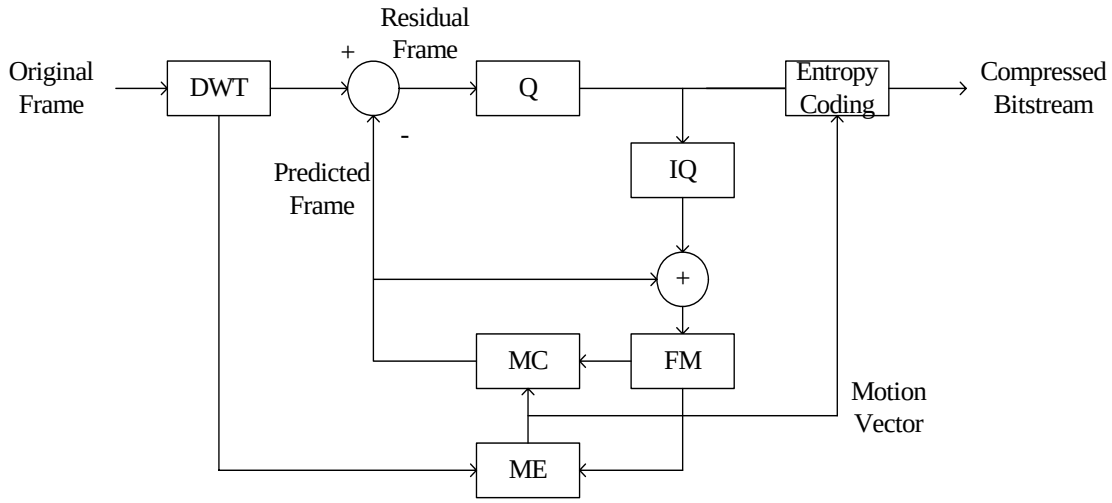
The number of bits used in this example using the SPECK algorithm is 440 bits if only one bit is used to identify the sign of significant coefficient in each pass. The

EZW algorithm uses 467 bits to encode the same matrix as that of the SPECK algorithm if the Huffman coding is employed to encode the dominant symbols generated in the dominant pass. As a result, the SPECK algorithm can achieve superior compression performance than the EZW algorithm and attains the reversible (lossless) coding. In addition, subsequent entropy coding, such as arithmetic coding, can achieve further compression after applied in the SPECK algorithm. Besides, it can achieve progressive transmission since the important information is sent to the decoder first.

2.7 Overview of the framework of the 2D wavelet video coder

As the Discrete Cosine Transform (DCT) suffers from "blocking effect" in low bit-rate applications, the Discrete Wavelet Transform (DWT) is used to eliminate the blocking artifacts due to its global decomposition of the entire image or video frame. Therefore, the DWT is widely used in image processing and video technology. The 2-dimensional (2D) wavelet video coder makes use of the concept of the traditional hybrid video-coding to remove both spatial and temporal redundancies. Figure 2.22 depicts the block diagram of the 2D wavelet video encoder. Firstly, the original video frames are performed DWT to remove the spatial redundancy. Then, the block-based motion estimation is carried out in the wavelet domain for the wavelet-transformed frames in order to remove the temporal redundancy of consecutive video frames. The motion vectors obtained are entropy encoded and transmitted to the decoder. During motion compensation, the predicted frame is obtained by the motion vectors and the decoded frame which is stored in the frame memory. After that, the original frame is subtracted by the predicted frame to form the residual frame. Finally, the residual frame is quantized, entropy encoded to form the encoded bitstream. Besides, the quantized

residual frame is inversely quantized to store in the frame memory for encoding next frame.



ME: Motion Estimation
MC: Motion Compensation
DWT: Discrete Wavelet Transform
Q: Quantization
IQ: Inverse Quantization
FM: Frame Memory

Figure 2.22 Block diagram of the framework of the 2D wavelet video encoder

The advantages of the DWT are that it is free from blocking artifacts, provides superior compression performance as compared to that of the DCT which is always used in traditional image and video coding systems. The DWT is also scalable in nature which can meet to different low-end display requirements and adapt different network conditions. Therefore, it is possible that the DWT will be used in the next generation image and video coding standards.

Besides, the motion estimation and compensation in the wavelet domain bear some similarity and difference as compared with the conventional video coding standards. Let us exploit some properties in the wavelet domain, such as the correlation between subbands across different levels inside the wavelet pyramid in order to enhance the speed of the motion estimation. In the following section, a classic wavelet-domain

motion estimation algorithm will be discussed and some modifications of this conventional algorithm will also be included.

2.8 Literature review of the wavelet-domain motion estimation and compensation in the 2D wavelet video coder

In this section, the typical motion estimation algorithm in wavelet domain will be mentioned in section 2.8.1. Some improvements of this conventional algorithm will be discussed in sections 2.8.2 and 2.8.3. The framework of the wavelet video coding system that will be discussed in the following sections is shown in Figure 2.22.

2.8.1 Multi-resolution Motion Estimation and Compensation (MRME)

A conventional motion estimation algorithm in the wavelet domain, Multi-resolution Motion Estimation (MRME) [74], is discussed in this section. The objectives of this approach are to reduce the searching time for motion estimation and provide a smooth motion vector field. As shown in Figure 2.23, a video frame is decomposed into many levels with different resolutions by the DWT with three levels. Although the motion activities inside a frame at different levels of the pyramid are different, they are highly correlated because they specify the same motion activity with different scales. According to this observation, the motion vectors of the LL, LH, HL and HH subbands at the lowest resolution level are calculated first. Then, the motion vectors obtained in the previous resolution level are used as an initial searching position of the current level

and the refinement is performed within a reduced search window in order to reduce the operations used in motion estimation. Figure 2.24 depicts the MRME scheme. $V_i(x, y)$ represents the motion vector with the centre (x, y) at level i. The motion vector of a LH subband, $V_i(x, y)$, which can be written as $V_i(x, y) = 2 \times V_{i+1}(x, y) + \Delta(x, y)$, for $i = 1$ or 2, where $\Delta(x, y)$ is the refinement motion vector. As an accurate initial searching position can be obtained, the search window for the refinement can be reduced in each level. For example, the search window in level 3 is ±15, it will be ±7 in level 2 and ±3 in level 1 (see Figure 2.23). As a result, the searching time and operations used in motion estimation can be reduced significantly.

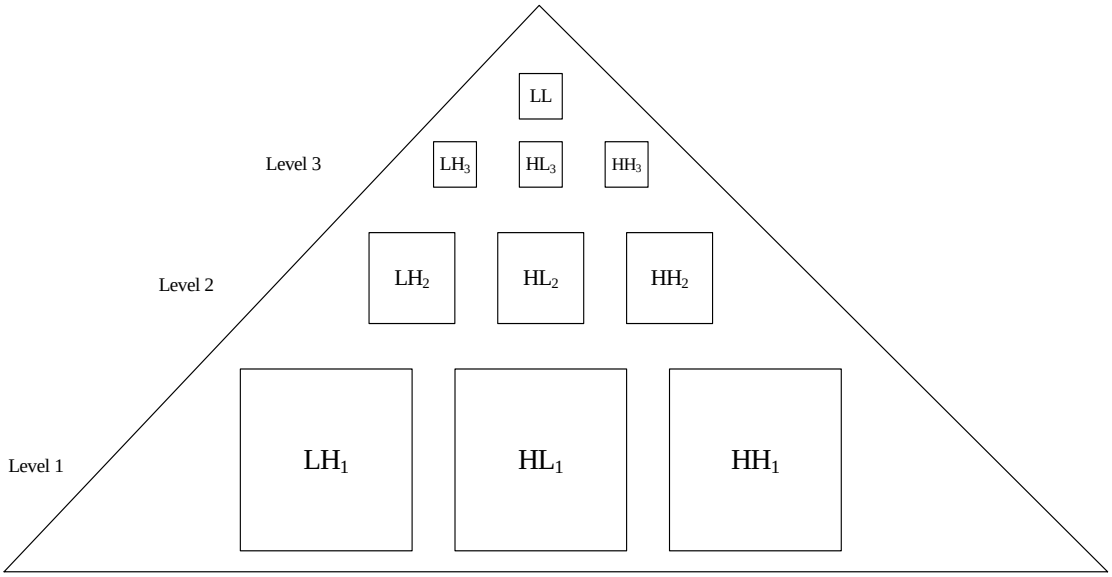


Figure 2.23 The pyramid structure of wavelet decomposition and reconstruction

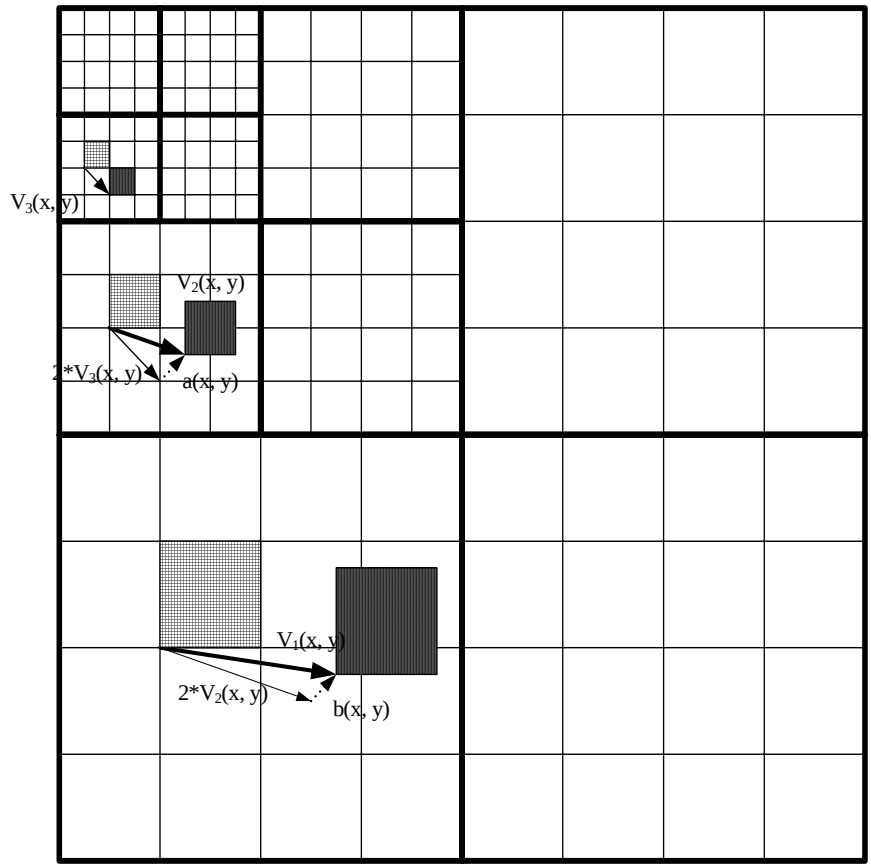


Figure 2.24 Variable block-size multiresolution motion estimation

The human visual system is more sensitive to the degradation in the low frequency components than that of the high frequency components. In other words, the human eye is more susceptible to the error in detail regions than that near the edges. Therefore, the block size is varied across different resolution levels. For example, the block size is 2 × 2 in level 3, 4 × 4 in level 2 and 8 × 8 in level 1. As a result, the number of blocks is kept constant at different resolutions. Under this situation, the motion of small objects in the lowest resolution level can be detected. If the block size remains unchanged for every level, two objects moving in different directions may be grouped into a block in the lowest resolution level. But only one direction can be kept tracked. Therefore, an inaccurate motion vector may be obtained leading to reduce the compression efficiency. The variable block-size approach is adopted in the MRME scheme in order to overcome this difficulty.

2.8.2 Adaptive MRME (AMRME), Bi-directional MRME (BMRME) and Fast MRME (FMRME)

The conventional MRME algorithm [74] is mentioned in the previous section. In this section, three modified algorithms [75] will be discussed. They can be combined together in order to improve the reconstructed quality and reduce the searching time.

2.8.2.1 Adaptive Thresholding Technique (AMRME)

The original MRME scheme [74] performs searching in all subbands at all levels but the correlation between the high frequency subbands of two consecutive video frames is not very high, so the prediction in high frequency subbands may be inaccurate. The uncorrelated high frequency subbands of successive frames are due to two reasons. The first one is that the DWT is translational invariant. That means, if the object in spatial domain moves to right by a pixel, it may not shift to right by one pixel in wavelet domain. Actually, it will shift to right by two pixels due to the dyadic decomposition. The other reason is that the high frequency frame is mainly composed by the edge information, so it will be changed rapidly even though the video sequence contains small motion.

According to this fact, if the absolute difference between the current block and the best match block is greater than a threshold value, the block will be discarded and a zero block will be initialized in the decoder. From the experimental result, the threshold value should be smaller than the energy of the current block. The threshold factor is defined as the ratio of the threshold value and the energy of the current

block. It should be between 0.6 and 0.9 in order to provide superior compression performance.

The AMRME [75] can reduce the bits used to encode the motion vectors because the motion vectors of the mismatch block will not be encoded. However, the computational complexity is higher than the conventional approach as the energy of the current block is required to calculate in order to find the threshold value.

2.8.2.2 Bi-directional Motion Estimation (BMRME)

In the traditional MRME scheme [74], the previous frame is always used as the reference frame but the best matched block may not be in the previous frame. Instead, it may be located in the future frame. So, the bi-directional prediction is employed in the MRME scheme [75] in order to improve the reconstruction quality. If the bi-directional motion estimation is performed in all subbands, the computational complexity will be increased significantly. Therefore, it is only performed in the subbands at the lowest resolution level, i.e. LL, LH_3, HL_3 and HH_3 subbands as shown in Figure 2.23, because the motion vectors are obtained hierarchically from low to high resolution levels. As a result, the quality of the reconstructed sequence can be improved and the computational complexity is only slightly increased.

2.8.2.3 Fast MRME (FMRME)

After decomposing the video frames by the DWT, motion estimation is performed to obtain the motion vectors in all subbands at all levels in the standard MRME approach [74]. The high frequency subbands at the same level are highly correlated to each other because they represent the same motion activities at the same

resolution. The wavelet coefficients of the high frequency subbands at the same level can be combined together to form a new subband, entitled as all-orientation subband [75], as depicted in Figure 2.25 and the motion estimation is performed in the all-orientation subband only in order to save the operations in motion estimation.

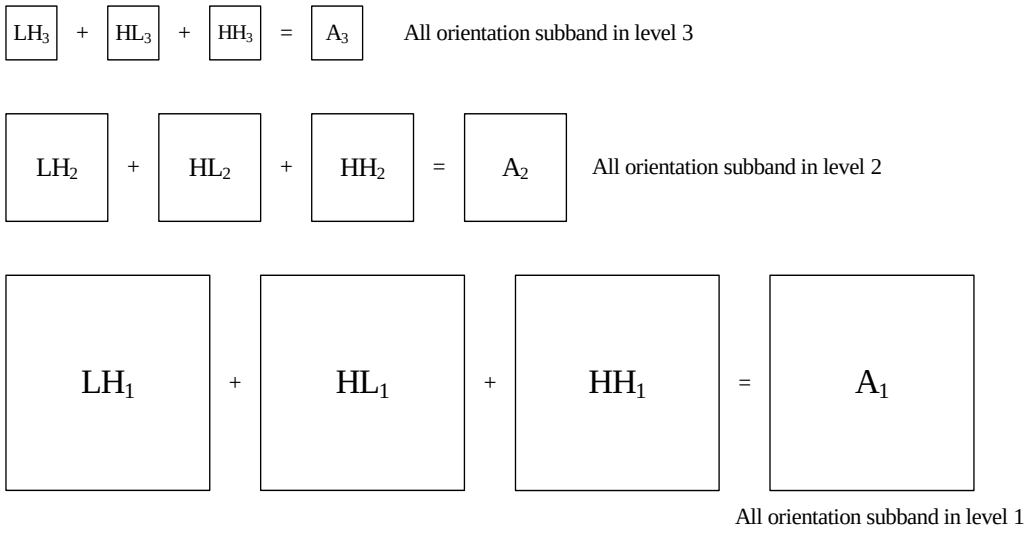


Figure 2.25 All orientation subbands in FMRME scheme

Although the FMRME scheme can reduce the searching time for motion estimation, it degrades the reconstruction quality. Therefore, the Adaptive Multiresolution Motion Estimation (AMRME), Bi-directional Multiresolution Motion Estimation (BMRME) and Fast Multiresolution Motion Estimation (FMRME) [75] are combined to each other in order to provide better quality and reduce the computational complexity for motion estimation.

2.8.3 Enhanced MRME (EMRME)

One more modified approach of the MRME scheme is discussed in this section. For the conventional MRME approach [74], all subbands at all levels are performed searching, so it is very computationally intensive. Due to the fact that a large portion of two consecutive frames is very similar to each other according to the large temporal

correlation between two successive frames, if we only carry out motion estimation in the potential motion area (PMA), in which the motion will be likely occurred, instead of all locations inside a subband, the operations for the searching can be reduced dramatically [76]. A binary mask, $M^i_j(x, y)$, ($i = 1, 2, 3$ *and* $j = 2, 4, 8$), is used to define whether a pixel located at (x, y) of a frame of subband i at level j is inside the PMA or not, where x and y are the horizontal and vertical positions of a frame respectively. If the absolute difference of the LL subbands of the reference block and the current block is greater than a threshold, T_0, $M_8(x, y)$, corresponds to the pixels inside LL subband is set to one. Otherwise, it is set to zero. As depicted in Figure 2.26, the corresponding positions in M^i_8 ($i = 1, 2, 3$) and that at the two higher resolution levels, i.e. M^i_4 and M^i_2 ($i = 1, 2, 3$), are also set to one. Besides, for point (x, y) at subband i at level j, where M^i_j is set to zero, if the absolute difference of the corresponding subband at two consecutive frames are greater than another threshold value, T_1, M^i_j is marked as one and propagate to the highest resolution level as described above if M^i_j is not located at the highest resolution level as illustrated in Figure 2.26.

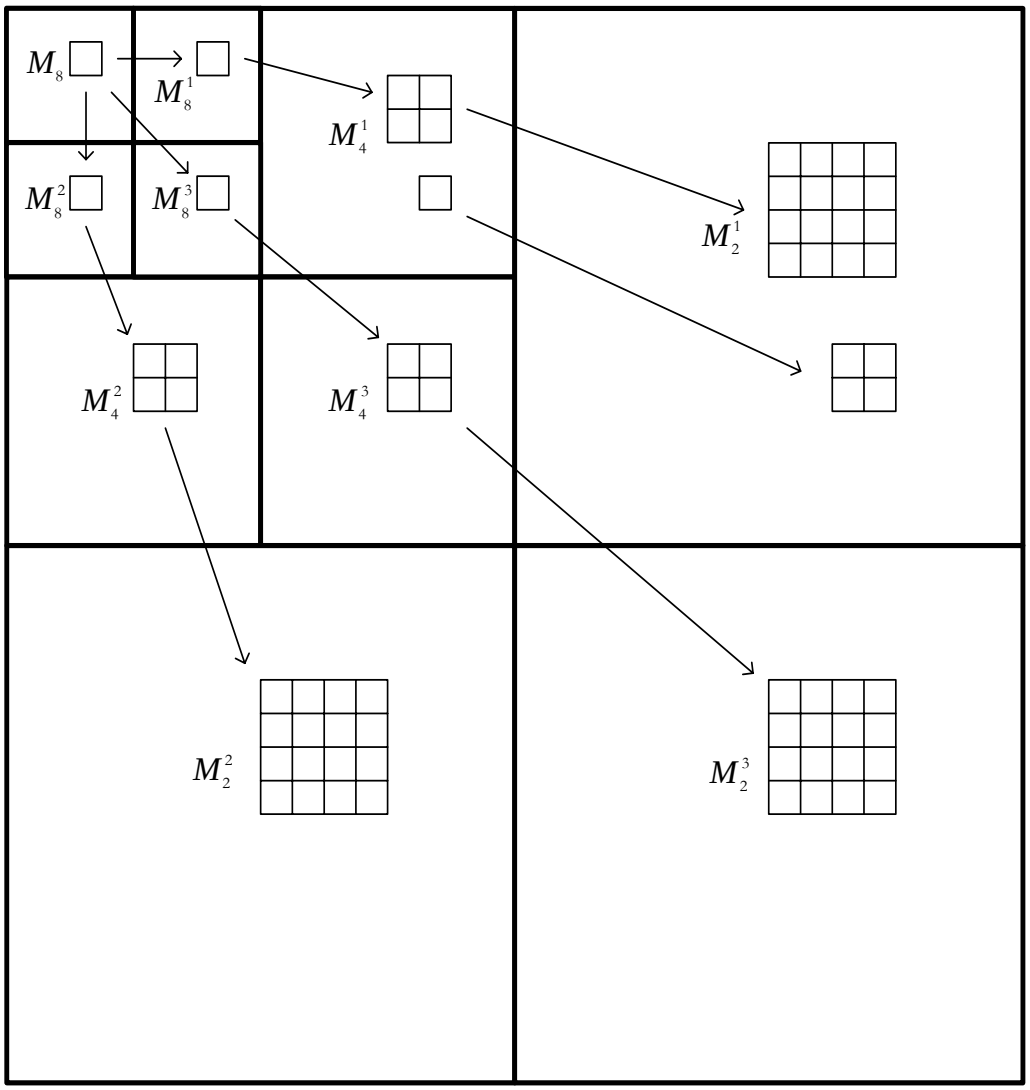


Figure 2.26 Mask propagation for the enhanced MRME (EMRME)

Actually, the Enhanced Multiresolution Motion Estimation (EMRME) [76] makes use of the zerotree structure to indicate the PMA in order to exploit the correlation between the subbands across different levels. Since motion estimation is only performed at PMA, which is usually a small part of the whole frame, the computational complexity can be reduced considerably.

2.9 Overview of the framework of the 3D wavelet video coder

As mentioned in section 2.8, the 2D wavelet video coder can achieve spatial scalability in order to adapt to different display devices such as the high-resolution display in desktop computer, middle-resolution display in laptop computer and low-resolution display in PDA device. However, it cannot attain the temporal scalability. Instead of using 2D wavelet transform, the 3 dimensional (3D) wavelet transform is performed to achieve both spatial and temporal scalabilities. The compressed bitstream of 3D wavelet video coder can be more robust to fluctuation of the network condition. If the network is very busy, only a portion of the bitstream will be transmitted to the decoder. Then, the decoder can reconstruct the video sequence in low frame rate such as half or a quarter of the original frame rate. On the contrary, if the bandwidth of the network is large enough to convey the whole encoded bitstream to the decoder, the decoder can work out the reconstructed sequence in full frame rate in order to achieve a high quality and resolution video.

Figures 2.27 (a) and (b) depict the block diagrams of the 3D wavelet video encoder and decoder respectively. The 3D wavelet transform is separated into two parts which are the 1D temporal DWT and 2D spatial DWT and they are carried out independently. At the encoder, the original video frames are partitioned into different Groups of Frames (GOF). Firstly, each GOF is performed the temporal wavelet transform which applies the 1D-DWT in temporal dimension in order to reduce the temporal redundancy in the consecutive video frames. Usually, the motion estimation is involved in the temporal wavelet transform in order to improve the compression efficiency and visual quality of the low frequency frames and this part will be discussed in section 2.10 in details. The temporal transformed frames are then carried out the 2D-

DWT in order to reduce the spatial redundancy in each frame. Then, quantization is applied on each wavelet-transformed frame to reduce the precision of the wavelet coefficients. Finally, the quantized transformed frames are carried out the entropy encoding in order to convert it into a bitstream and send it to the decoder. At the decoder, it receives the encoded bitstream and performs the reverse operations in the encoder to reconstruct the video sequence as illustrated in Figure 2.27(b).

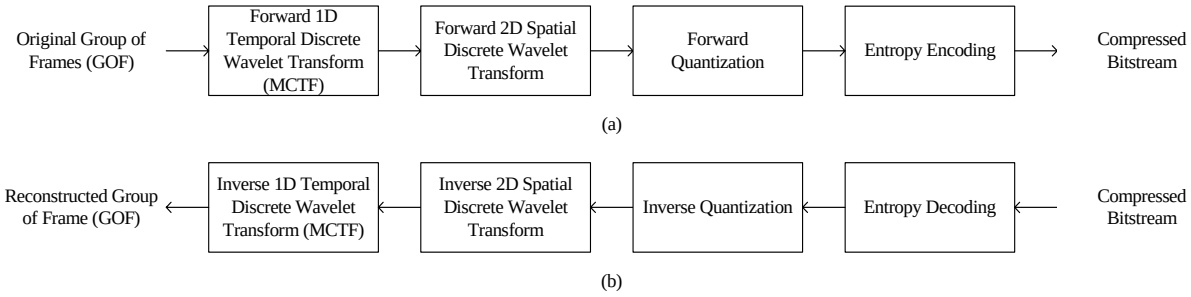


Figure 2.27 The block diagram of the 3D wavelet video codec

2.10 Literature review of the Motion Compensated Temporal Filtering (MCTF)

The 3D-DWT [52] is used in wavelet video codec instead of the 2D-DWT in order to achieve additional compression performance and both spatial and temporal scalabilities. Usually, the 3D-DWT is executed in separate. The 1D temporal DWT is carried out before the 2D spatial DWT. The temporal 1D-DWT decomposes the video frames in temporal dimension. If the temporal filtering without involving the motion compensation, it will produce the ghosting artifacts in the low frequency frame leading to reduce the visual quality of the low frequency frame. It is because the low frequency frame is obtained by the high frequency frame. If the high frequency frame contains error, the error will be added into the low frequency frame to introduce the ghosting

artifacts. For the reduced frame rate application, i.e. half or a quarter of the original frame rate, only low frequency frames are reconstructed and displayed in the decoder. The ghosting artifacts will degrade the visual quality of the low frequency frame. Therefore, the motion estimation and compensation is usually adopted in the temporal filtering, i.e. motion compensated temporal filtering (MCTF) [52]. After involving the motion estimation inside both predict and update steps, the high frequency frame represents the error of prediction. If the motion model can accurately capture the motion of the video sequence, the prediction error will be small so that the coding efficiency can be improved. Besides, the ghosting artifacts in the low frequency frame can be eliminated because it represents the high quality reduced frame-rate video after performing the temporal decomposition. The transform does not introduce the ghosting artifacts into the low frequency frame. As a result, its visual quality is comparable to the temporally down-sampled original video frame. In sections 2.10.1 and 2.10.2, two kernels, which are the Haar and Bi-orthogonal 5/3 kernels respectively, are used as the examples to illustrate the concept of the MCTF [52].

2.10.1 Haar kernel

When Haar kernel is used, the low frequency frame, $l[m, n]$, and high frequency frame, $h[m, n]$, are achieved by the following equations without motion compensation where $x_1[m, n]$ and $x_2[m, n]$ are two frames from the video sequence.

$$h[m,n] = \frac{1}{2}\left(x_2[m,n] - x_1[m,n]\right)$$
$$l[m,n] = x_1[m,n] + h[m,n]$$

When motion compensation is included before carrying out temporal decomposition, the lifting steps are modified as follows.

$$h[m,n] = \frac{1}{2}\left(x_2[m,n] - \left(W_{1-2}(x_1)\right)[m,n]\right)$$
$$l[m,n] = x_1[m,n] + \left(W_{2-1}(h)\right)[m,n]$$

$W_{1\text{-}2}$ represents the motion compensated mapping of the first frame onto the coordinate system of the second frame, so that $(W_{1\text{-}2}(x_1))[m, n] \approx x_2[m, n]$, $\forall(m, n)$. $W_{2\text{-}1}$ represents the motion compensated mapping of the second frame onto the coordinate system of the first. When there is no motion, $W_{1\text{-}2}$ and $W_{2\text{-}1}$ are both the identity operators. $W_{1\text{-}2}$ and $W_{2\text{-}1}$ are not generally inverses of one another. The former represents the backward motion field while the latter represents the forward motion field. When the scene motion is neither expansive nor contractive, i.e. pure translation, skewing or rotation, $W_{1\text{-}2}$ and $W_{2\text{-}1}$ are indeed inverses of one another. After performing the temporal decomposition, the coefficients of the high frequency frame are close to zero because the high frequency frame represents the residual frame and its energy has a direct impact on the coding gain. The low frequency frame represents the original frame in the video sequence and it is free from ghosting artifacts. Besides, most energy is concentrated on it.

2.10.2 Bi-orthogonal 5/3 kernel

In the Haar kernel, we always use the previous frame as the reference frame. However, the best match of the current block may not be located in the previous frame only. Instead, one of the best predictions is to make use of both the previous and future frames. For the Bi-orthogonal 5/3 kernel, two reference frames, which are the previous and future frames, are used.

$$h_k[m,n] = x_{2k+1}[m,n] - \frac{1}{2}\left(x_{2k}[m,n] + x_{2k+2}[m,n]\right)$$
$$l_k[m,n] = x_{2k}[m,n] + \frac{1}{4}\left(h_{k-1}[m,n] + h_k[m,n]\right)$$

The above equations show the case without motion compensation while the following equations show the case with motion compensation.

$$h_k[m,n] = x_{2k+1}[m,n] - \frac{1}{2}\left(W_{2k,2k+1}(x_{2k})[m,n] + W_{2k+2,2k+1}(x_{2k+2})[m,n]\right)$$
$$l_k[m,n] = x_{2k}[m,n] + \frac{1}{4}\left(W_{2k-1,2k}(h_{k-1})[m,n] + W_{2k+1,2k}(h_k)[m,n]\right)$$

$W_{k1,k2}$ denotes the motion compensated mapping from frame k_1 onto the coordinate system of frame k_2. Similar to the Haar kernel, the high frequency frame represents the residual from a bi-directional motion compensated prediction of the relevant odd indexed original video frame. If the motion model can capture the motion of the video sequence accurately, the coefficients in the high frequency frame tend to zero.

The bi-orthogonal 5/3 kernel yields better performance than the Haar kernel due to the bi-directional prediction in both predict and update steps. As the motion compensation is introduced into the lifting steps, so the coding efficiency is improved for both kernels. Due to the absence of ghosting artifacts in the low frequency frame, the visual quality of the low frequency frame can be increased. As a result, the temporal scalability can be achieved by partial reconstruction of the temporal filtering, i.e. by dropping the high frequency frame.

2.11 Modifications of the MCTF

In the previous section, the conventional MCTF scheme [52] was mentioned. According to this classical approach, some modifications will be discussed in the following sections. Due to the dyadic decomposition of the wavelet transform used in both 1D temporal DWT and 2D spatial DWT, the spatial and temporal scalabilities can only achieve a factor of two. Some variations of the MCTF approach in dyadic scheme are described in section 2.11.1. Besides the dyadic scheme, the MCTF scheme with the

temporal scalability of a power of three will be stated in section 2.11.2. This three-band scheme can make the encoded bitstream adapt to different network conditions.

2.11.1 Dyadic Scheme

In this section, the dyadic decomposition is used in the temporal filtering and some modifications of the MCTF are revealed.

2.11.1.1 Optimization of predict operator

Section 2.10.2 mentions the conventional MCTF scheme [57] using the bi-orthogonal 5/3 kernel which uses the previous and future frames as the reference frames in order to reduce the bi-directional prediction error in the high frequency frame leading to an increase in coding gain. Figure 2.28 illustrates the formation of the high frequency frame graphically. The following equation demonstrates the calculation of the high frequency frame, where $x_{2t+1}(n)$ is the pixel at spatial location *n* in frame *2t+1*, $v^{+}_{2t+1}(n)$ is the forward motion vector to predict the frame *2t+1* from the frame *2t*, $v^{-}_{2t+1}(n)$ is the backward motion vector to predict the frame *2t+1* from the frame *2t+2* and $h_t(n)$ is the high frequency frame.

$$h_t(n) = x_{2t+1}(n) - \frac{1}{2}\left\{x_{2t}\left[n - v^{+}_{2t+1}(n)\right] + x_{2t+2}\left[n - v^{-}_{2t+1}(n)\right]\right\}$$

This equation represents that the pixel *n* finds the best match locations in frames *2*t and *2t+1,* i.e. *p* and *q* respectively, where *p* and *q* are denoted by $n - v^{+}_{2t+1}(n)$ and $n - v^{-}_{2t+1}(n)$ in frames *2t* and *2t+2* respectively. As the prediction does not have any preference on the reference frames, so the weighting factor is a half. According to the

above equation, the high frequency frame corresponds to the residual signal of the bi-directional prediction and it contributes to a deep impact on the compression performance. Therefore, two motion vectors, $\left(\hat{v}_{opt}^{+}, \hat{v}_{opt}^{-}\right)$, are obtained by minimization the distortion function, *d*, as shown below, where W^{+} and W^{-} are the search windows in the frames *2t* and *2t+2* respectively.

$$\left(\hat{v}_{opt}^{+}, \hat{v}_{opt}^{-}\right) = \arg\min_{\substack{v^{+}\in W^{+} \\ v^{-}\in W^{-}}} \sum_{n} d\left[x_{2t+1}(n) - \frac{1}{2}\left\{ x_{2t}\left[n - v_{2t+1}^{+}(n)\right] + x_{2t+2}\left[n - v_{2t+1}^{-}(n)\right]\right\}\right]$$

As two motion vectors, $\hat{v}^{+}$ and $\hat{v}^{-}$, are involved in the high frequency frame, the forward and backward motion estimations are carried out separately in conventional approach as depicted in the following equations.

$$\begin{cases} \hat{v}^{+} = \arg\min_{v^{+}\in W^{+}} \sum_{n} \left[d\left(x_{2t+1}(n) - x_{2t}\left(n - v_{2t+1}^{+}(n)\right)\right)\right] \\ \hat{v}^{-} = \arg\min_{v^{-}\in W^{-}} \sum_{n} \left[d\left(x_{2t+1}(n) - x_{2t+2}\left(n - v_{2t+1}^{-}(n)\right)\right)\right] \end{cases}$$

The conventional approach can only minimize the distortion for the forward and backward motion estimations independently but not for the high frequency frame. In order to optimize the predict operator, the procedures of the modified predict operator [103] are summarized as follows.

Step 1) Find the forward motion vector, $\hat{v}_{1}^{+}$.

$$\hat{v}_{1}^{+} = \arg\min_{v^{+}\in W^{+}} \sum_{n} \left[d\left(x_{2t+1}(n) - x_{2t}\left(n - v_{2t+1}^{+}(n)\right)\right)\right]$$

Step 2) Calculate the backward motion vector, $\hat{v}_{2}^{-}$, by using the forward motion vector, $\hat{v}_{1}^{+}$, found in the previous step.

$$\hat{v}_{2}^{-} = \arg\min_{v^{-}\in W^{-}} \sum_{n} \left[d\left(x_{2t+1}(n) - \frac{x_{2t}\left(n - \hat{v}_{1}^{+}(n)\right) + x_{2t+2}\left(n - v_{2t+1}^{-}(n)\right)}{2}\right)\right]$$

Step 3) Update the forward motion vector, $\hat{v}_3^+$, by using the backward motion vector, $\hat{v}_2^-$, obtained in step 2. Then, the optimum forward and backward motion vector fields are the $\hat{v}_3^+$ and $\hat{v}_2^-$ respectively.

$$\hat{v}_3^+ = \arg\min_{v^+ \in W^+} \sum_n \left[d\left(x_{2t+1}(n) - \frac{x_{2t}\left(n - v_{2t+1}^+(n)\right) + x_{2t+2}\left(n - \hat{v}_2^-(n)\right)}{2} \right) \right]$$

As the error of the bi-directional prediction can be reduced by obtaining the optimum forward and backward motion vectors, so the coding efficiency can be improved.

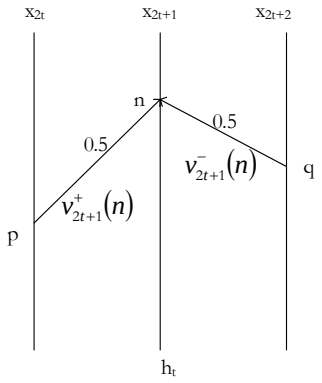


Figure 2.28 Predict operator of the MCTF scheme using the Bi-orthogonal 5/3 kernel

2.11.1.2 Skipping of update operator

The previous section discusses an improved method to obtain the high frequency frame in order to reduce the prediction error. Therefore, the coding efficiency can be increased. The high and low frequency frames are calculated by the following equations, where h_t and l_t denote the high and low frequency frames respectively. After finding the high frequency frame, the current and previous high frequency frames are used to obtain the low frequency frame as depicted in Figure 2.29 and Figure 2.30.

$$h_t(n) = x_{2t+1}(n) - \frac{1}{2}\left\{x_{2t}\left[n - v_{2t+1}^+(n)\right] + x_{2t+2}\left[n - v_{2t+1}^-(n)\right]\right\}$$

$$l_t(p) = x_{2t}(p) - \frac{1}{4}\left\{h_{t-1}\left[p + v_{2t-1}^-(m)\right] + h_t\left[p + v_{2t+1}^+(n)\right]\right\}$$

If the motion model can capture the motion of the video sequence accurately, the wavelet coefficients of the high frequency frame will tend to zero according to the above equations. Since the high frequency frame is used to obtain the low frequency frame, so the terms, $x_{2t}(p)$, is much greater than the prediction error. Therefore, the low frequency frame, l_t, is similar to frame *2t*, x_{2t}. On the contrary, if the model fails to keep track the motion of the video, the compression efficiency will be adversely affected. Besides, as the high frequency frames are used to attain the low frequency frame, so some errors will be added into the low frequency frame leading to the introduction of the ghosting artifacts. As a result, the visual quality of the low frequency frame will be degraded. In order to reduce the ghosting effect in the low frequency frame, the update operator of the low frequency frame is modified as follows [104].

$$l_t(p) = x_{2t}(p)$$

According to the above equation, the low frequency frame is the even frames in the original video sequence. Therefore, the visual quality of the low frequency frame can be assured. In addition, this operation is used instead of the low-pass filtering and downsampling so that the computational complexity can be reduced.

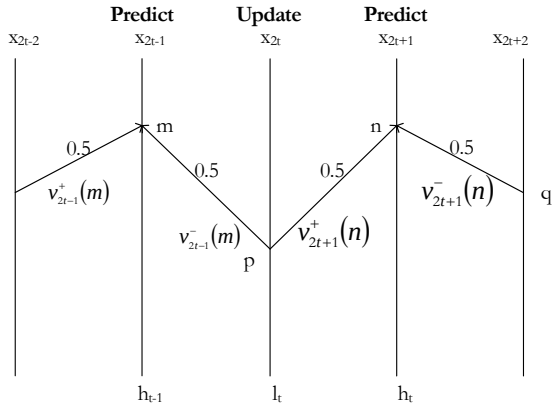


Figure 2.29 Predict and update operators of the Bi-orthogonal 5/3 kernel

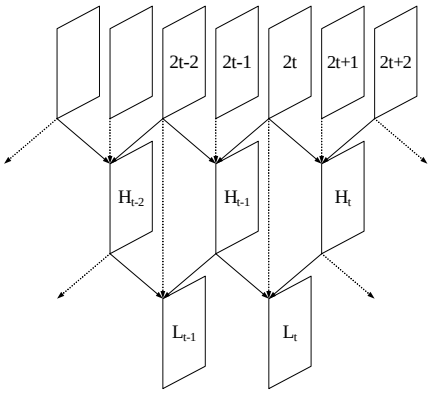


Figure 2.30 Temporal decomposition using the Bi-orthogonal 5/3 kernel

2.11.1.3 Temporal prediction and differential coding of motion vectors

The conventional MCTF framework [52] obtains the forward and backward motion vector fields in each temporal level independently in order to remove the temporal correlation between consecutive video frames in the same temporal level. Then, the obtained motion vectors are entropy encoded and transmitted as side information to the decoder. Besides of the temporal correlation between video frames, there exists a large correlation of the low frequency frames among different temporal levels. If such relationship is used, the number of bits used to encode the motion information can be reduced [56]. The bi-orthogonal 5/3 kernel involves two reference frames for bi-directional prediction so that the number of motion vectors needs to be encoded is larger as compared with that of the Haar kernel which uses only one reference frame for prediction. Thus, the motion vector prediction method for bi-orthogonal 5/3 kernel is discussed in the following section.

Figure 2.31 depicts the one level temporal decomposition for the bi-orthogonal 5/3 kernel which involves bi-directional prediction in both predict and update operators. For finding the high frequency frame, motion estimation is performed between the

frame 1 and frame 0, and frame 1 and frame 2, where frames 0 and 2 are the reference frames to obtain the motion vector fields, MV1 and MV2 respectively. After obtaining the high frequency frame, it is used to calculate the low frequency frame. If further temporal decomposition is performed, motion estimation is performed between the consecutive low frequency frames in order to attain the motion vector field, MV3. Traditionally, these motion vector fields are found independently. However, the low frequency frames among different temporal levels are highly correlated. Therefore, MV1 and MV2 can be used to obtain MV3 as shown in Figure 2.32. Firstly, the initial searching position is estimated by subtracting MV1 from the MV2, i.e. MV1 – MV2. Then, the search window is shifted by the initial searching point to obtain the refinement motion vector. Finally, the resultant motion vector, MV3, is found by adding the initial predicted motion vector, i.e. MV1 – MV2, and the refinement motion vector. As the motion vector fields, MV1 and MV2, have already been entropy encoded, so only the small refinement motion vector requires to be encoded and sent to the decoder in order to save the number of bits used to encode the motion vectors.

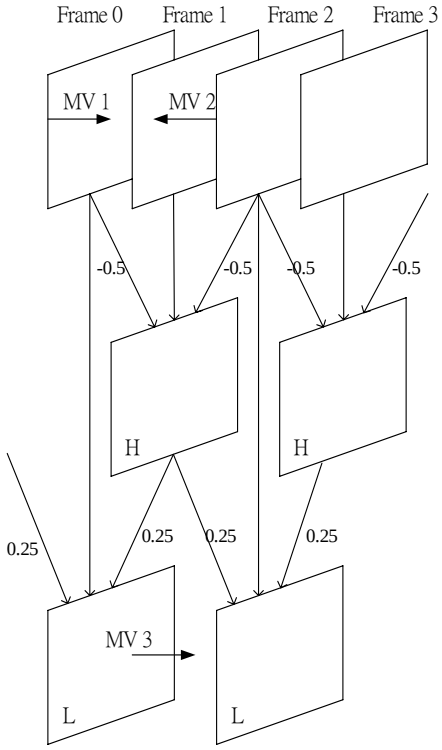


Figure 2.31 One level MCTF with bi-directional 5/3 kernel using lifting structure

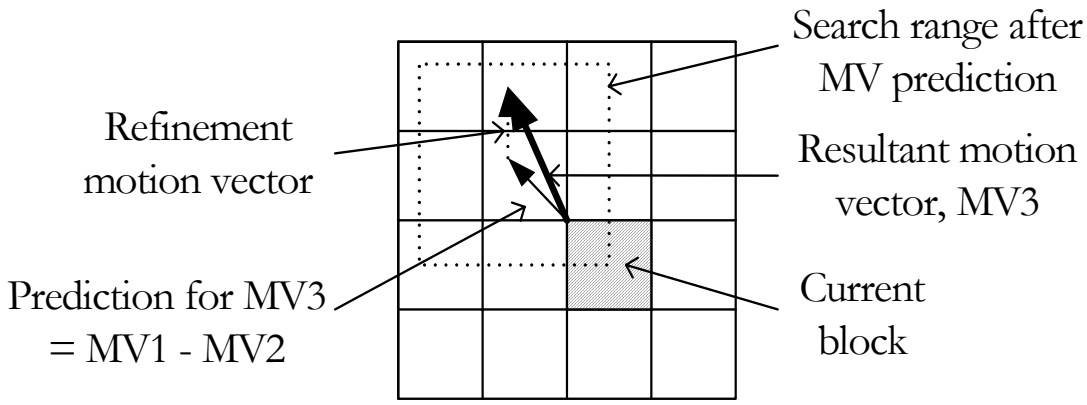


Figure 2.32 Motion vector prediction for the estimation of MV3

2.11.2 Three-Band Scheme

The section 2.11.1 describes the MCTF framework under the dyadic wavelet decomposition in the temporal direction. The major disadvantage of the dyadic scheme is that it can only achieve a temporal scalability of a power of two. In this section, the modified MCTF framework, three-band scheme, with the temporal scalability of a

power of three is mentioned in order to make the encoded bitstream adapt to the fluctuation of the network condition. Sections 2.10.2.1 and 2.10.2.2 discuss the three-band scheme for the Haar and Bi-orthogonal 5/3 kernels respectively.

2.11.2.1 Haar kernel

The three-band scheme using Haar kernel [99] is mentioned in this section. As it only uses one reference frame to obtain the high frequency frame, so it is considered to be the Haar kernel. The conventional MCTF framework [52] uses the previous frame as the reference frame to calculate the high frequency frame in the predict operation. The three-band scheme finds two high frequency frames, which are forward and backward high frequency frames, by using the previous and future frames as reference frames in the forward and backward predict operators respectively as depicted in Figure 2.33 and Figure 2.34. Since motion estimation is involved in the lifting steps, so the forward and backward motion vectors, i.e. v^{+}_{3t+1} and v^{-}_{3t-1}, are obtained in order to perform motion compensation before the temporal filtering. After that, these two high frequency frames are used to compute the low frequency frame by the following equations.

$$h_t^{+}(n) = x_{3t+1}(n) - x_{3t}\left(n - v^{+}_{3t+1}\right)$$
$$h_t^{-}(m) = x_{3t-1}(m) - x_{3t}\left(m - v^{-}_{3t-1}\right)$$
$$l_t(p) = x_{3t}(p) + \frac{1}{4}\left[h_t^{+}\left(p + v^{-}_{3t+1}\right) + h_t^{-}\left(p + v^{+}_{3t-1}\right)\right]$$

Similar to the traditional MCTF scheme, the high frequency frame in the three-band scheme still represents the error of prediction. Two high frequency frames are discarded in order to achieve the temporal scalability of a power of three. Since the low frequency frame is found by the bi-directional update step, i.e. using two high frequency frames, so its visual quality is less affected by the ghosting artifacts as compared with that of the standard MCTF scheme with Haar kernel.

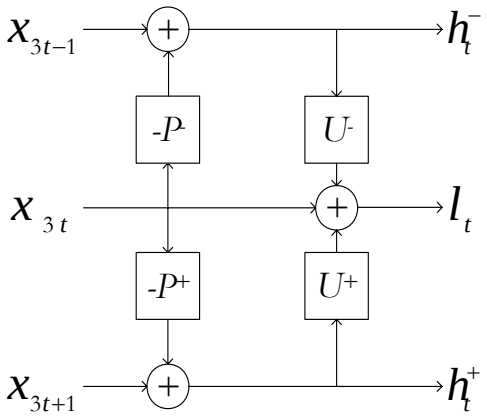


Figure 2.33 Predict and update steps of the three-band scheme for Haar kernel

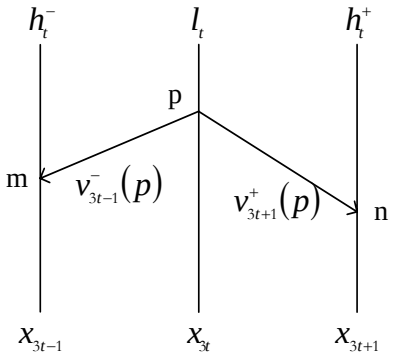


Figure 2.34 Predict operator of the three-band scheme for Haar kernel

2.11.2.2 Bi-orthogonal 5/3 kernel

The three-band scheme can further be extended to a more complex kernel such as the bi-orthogonal 5/3 kernel [100] in order to attain higher compression efficiency due to the longer filter length. Similar to the conventional MCTF framework, the predict operator makes use of the two reference frames, which are the previous and future frames, for motion estimation as illustrated in Figure 2.35. The forward and backward motion estimations are performed to obtain the forward and backward motion vector fields, v_{3t+1}^+ and v_{3t+1}^-, respectively. Then, the forward high frequency frame, h_t^+, is calculated by the previous and future frames, x_{3t} and x_{3t+2}, as reference frames by the

following equations, where β is the weighting factor of the reference frames. Similarly, the backward high frequency frame, h_t^-, is attained in the same way. Subsequently, the low frequency frame is found by these two high frequency frames in the bi-directional update operator.

$$h_t^+(n) = x_{3t+1}(n) - \beta x_{3t+2}\left(n - v_{3t+1}^-\right) - (1-\beta)x_{3t}\left(n - v_{3t+1}^+\right)$$
$$h_t^-(m) = x_{3t-1}(m) - \beta x_{3t-2}\left(m - v_{3t-1}^+\right) - (1-\beta)x_{3t}\left(m - v_{3t-1}^-\right)$$
$$l_t(p) = x_{3t}(p) + \frac{1}{4}\left[h_t^+\left(p + v_{3t+1}^+\right) + h_t^-\left(p + v_{3t-1}^-\right)\right]$$

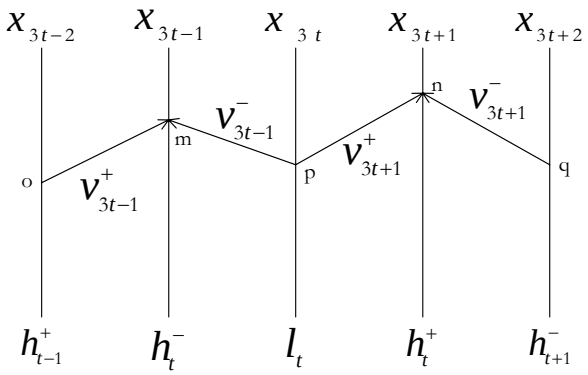


Figure 2.35 Predict and update operators of the three-band scheme for Bi-orthogonal 5/3 kernel

2.12 Conclusion Remarks

For wavelet video coding, the 2D-DWT is applied to each video frame to exploit the spatial redundancy. The motion estimation is adopted in the wavelet video encoder to remove the temporal redundancy in the successive frames and is usually performed in the wavelet domain by making use of the correlation of corresponding subbands among different decomposition levels. Thus, the computational complexity of motion estimation can be reduced significantly. However, it still consumes most of the execution time during encoding process. According to the observation that the wavelet coefficients having similar matching errors tend to be clustered to each other and this

correlation exists among different levels in the wavelet pyramid, the backward row-based Clustered Pixel Matching Error for Partial Distortion Search (backward CPME-PDS) is proposed to further reduce the number of operations for the motion estimation in the wavelet domain by exploiting the cross-correlation between corresponding subbands in the wavelet pyramid (more details can be found in chapter three).

Nevertheless, the 2D-DWT can only achieve the spatial scalability but not temporal scalability. Hence, the 3D-DWT is employed in the wavelet video coder to attain both temporal and spatial scalabilities. The 3D-DWT is carried out individually. First, the 1D-DWT is performed in the temporal direction to eliminate the relationship between consecutive video frames. Second, the 2D-DWT is executed in each frame to remove the spatial correlation in the frame. Usually, the motion estimation and compensation are applied in the temporal 1D-DWT in order to enhance the coding efficiency and improve the visual quality of the low frequency frames. Also, the computational effort of motion estimation is a major problem in the 3D wavelet video encoder. Due to the fact that there exists a large correlation among the successive frames and the wavelet transformed frames between different temporal levels, the median and cross-level motion vector prediction algorithm is proposed to improve the speed of motion estimation by exploiting such correlation (more details can be found in chapter four).

The Embedded Zerotree Wavelet (EZW) algorithm is commonly used to encode the wavelet coefficients in the wavelet image and video coder. A modified EZW algorithm with minimum weight and difference subband approach is proposed to improve the compression efficiency by discarding less important information and retaining the same visual quality as the conventional EZW algorithm (more details can be found in chapter five). The proposed algorithm can be extended to the Set-

Partition Embedded Block Coder (SPECK) algorithm and can be used in both 2D and 3D wavelet video coders.

Chapter 3

Motion estimation algorithm in the wavelet domain of the 2D wavelet video coder

3.1 Introduction

The Discrete Wavelet Transform (DWT) has received much attention recently due to its superior performance by comparing to the conventional block-based hybrid video coding such as Discrete Cosine Transform (DCT). It is well-known that the DCT produces the “blocking effect” in the low bit-rate applications. However, DWT is free from blocking artifacts by distributing the errors over the whole frame due to its nature of global decomposition. Besides, according to its multi-resolution nature in the wavelet pyramid, it can represent an image or video sequence flexibly and adapt to different bit-rates across the networks with different traffic situations.

The multiresolution nature in DWT is suitable for multi-resolution applications such as DTV / HDTV. Besides, when a video sequence is transmitted to the low-end display units such as mobile phone and PDA, the images of required resolution should be derived before displaying.

In this chapter, we propose a wavelet-based CPME-PDS algorithm by using the characteristic of clustered pixel matching errors in the hierarchical structure of the wavelet pyramid. The multiresolution / multifrequency nature of the DWT is an efficient tool to represent images and video signals for compression and transmission. The DWT decomposes a video frame into a set of subframes with different resolutions in subbands. This multiresolution nature provides a hierarchical representation of a

video frame. It provides useful information for us to develop efficient motion estimation algorithms. Hui, Siu and Chan [108] showed that pixel matching errors with similar magnitude tend to appear in a cluster in natural video sequences in the spatial domain. According to our observation, this clustering property is also found in the wavelet domain. We use this property and the hierarchical structure of the wavelet pyramid to develop an adaptive PDS applying in the MRME scheme. We create an adaptive index set based on this clustering characteristic in the highest resolution subband. Due to the hierarchical architecture of the wavelet pyramid, the index set can be down-sampled and re-numbered in other lower resolutions subbands. Hence, the required operations of finding the index set in each subband can be reduced. When the SEA is applied in the proposed algorithm, the speed of the motion estimation can further be improved by rejecting the searching positions in the search window for very slow motion video sequences.

Experimental results (Table 3.4) show that our proposed algorithm has a speed-up in motion estimation comparing to Full Search Algorithm (FSA) and conventional PDS using the MRME scheme. The proposed method can be used to enhance the efficiency of the motion estimation in the wavelet domain for the working schemes [74] - [78]. Furthermore, the high quality video conferencing and high quality documentary applications can benefit from the proposed method due to its superior performance in the slow motion video sequences. Besides, the encoding time is decreased by comparing to the conventional approach. Hence, the time delay for video conferencing and video surveillance applications can also be reduced, say for example. For the video surveillance application, as the encoding delay is reduced, the follow-up actions can be taken place for emergency events.

The organization of this chapter is shown as follows. The characteristic of the clustered pixel matching error in wavelet domain is analysed in Section 3.2. Details of the proposed algorithm is introduced in Section 3.3. Some experimental results are discussed in Section 3.4 and a brief conclusion is drawn in Section 3.5.

3.2 The characteristic of clustered pixel matching error in the wavelet domain

Ref. [108] shows that pixel matching errors tend to cluster together during motion estimation. Because the LL subband is a lower resolution version of an original image, the CPME-PDS in [108] can be applied directly.

It is found that the same property also exists in wavelet domain. Figure 3.1 gives an example of a real case which is used to explain this property. Figure 3.1 (a) depicts the matching of a one dimensional (1-D) block in LH subband (thick continuous line) within a 1-D searching window (thin dotted line). The corresponding pixel absolute matching errors also appear in a cluster form as shown in Figure 3.1 (b). The clustered pixel matching error characteristic can be used to improve the motion estimation efficiency in wavelet domain.

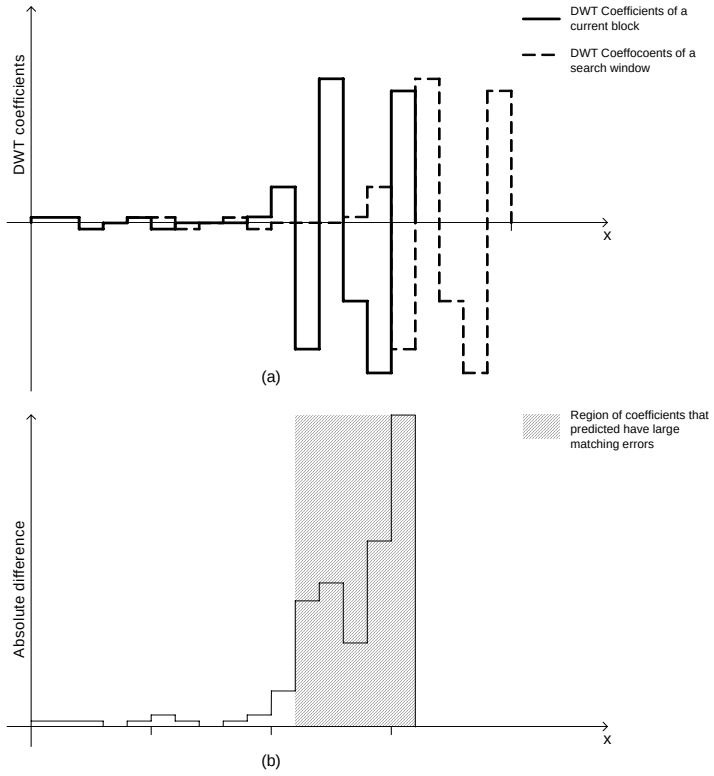
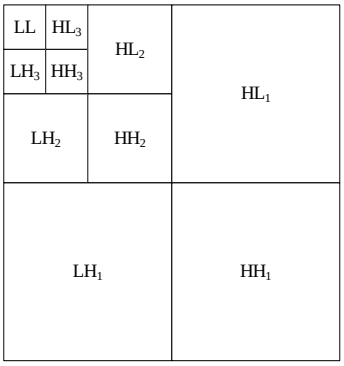
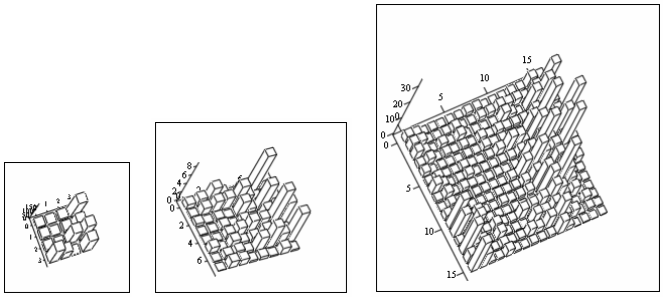


Figure 3.1 Matching of a one dimensional (1-D) block in LH subband within a 1-D searching window. (b) Corresponding pixel absolute matching errors of the target block at the current position

This clustering phenomenon in wavelet domain is demonstrated in Figure 3.2. Figure 3.2 (a) depicts the DWT hierarchical structure in wavelet pyramid. Figure 3.2 (b) illustrates the error blocks between a reference block and a current block in the LH_3, LH_2 and LH_1 subbands at the 9^{th} and 10^{th} frames of the video sequence "Akiyo" respectively. The error block of subband LH_1 clearly shows the clustering property in wavelet domain. Comparing the similarity between the error blocks of the subbands, we also find that this clustering property is highly correlated in each subband.



(a)



(b)

Figure 3.2 (a) DWT hierarchical structure in wavelet pyramid (b) Error blocks between a reference block and a current block in the LH_3, LH_2 and LH_1 subbands at the 9^{th} and 10^{th} frames of the video sequence "Akiyo" respectively

According to this analysis, we can determine that the CPME-PDS can be applied in the MRME scheme. Furthermore, an adaptive index set found in a subband at the highest resolution can be utilized in the corresponding lower resolution subbands. Hence, the redundant calculations can be avoided.

3.3 Proposed fast motion estimation algorithm in the wavelet domain of the 2D wavelet video coder

The row-based CPME-PDS is used to improve motion estimation in the MRME scheme. From the above simple analysis, the counting sort is not required to perform at each level. In our algorithm, we use the zero motion vector as the motion predictor to determine the adaptive index set for the highest resolution level. The index set result obtained in the previous operations is used to predict the sorting order of other levels. The proposed algorithm is summarized as follows.

Motion estimation of a block in the LL subband

Step 1)Use the zero motion vector as the motion predictor to calculate a reference value, *m*

$$m = \frac{1}{M \times N} \sum_{j=0}^{N-1} \sum_{i=0}^{M-1} I_{t-1}(x+i, y+j),$$

where *N* and *M* are the height and width of the block respectively

I_{t-1} is the intensity level of frame *t-1*

(*x*, *y*) is the coordinates of current block

Note that for the CIF format, $M = N = 4$ for level 3 and $M = N = 16$ for level 1 and for the QCIF format, $M = N = 2$ for level 3 and $M = N = 8$ for level 1 in our experiments

Step 2) Calculate the expected absolute pixel matching error in a row, $\left|p_{\exp}(n)\right|$, in the targeted block

$\left|p_{\exp}(n)\right| = \sum_{i=0}^{M-1} \left|I_t(i, k_n) - m\right|$, where k_n is the index of each row in the targeted block and n =0, …, *N-1*.

Note that $I_t(i, k_n)$ and *m* are floating point numbers and $p_{exp}(n)$ is truncated to integer for the sake of lower complexity.

Step 3) Use counting sort to obtain an adaptive index set, *S*, by sorting the expected absolute pixel matching error in a row, $\left|p_{\exp}(n)\right|$, in descending order, i.e. $\left|p_{\exp}(0)\right| \geq \ldots \geq \left|p_{\exp}(n)\right| \geq \ldots \geq \left|p_{\exp}(N-1)\right|$ with $S = \{k_n | n = 0,..., N-1\}$.

Step 4) Apply the adaptive index set, *S*, to calculate the partial Sum of Absolute Difference (SAD) in following equation during the searching of an outward spiral scanning.

$$SAD_p(x, y; u, v) = \sum_{j=0}^{p} \sum_{i=0}^{M-1} \left| I_t(x+i, y+k_n) - I_{t-1}(x+i+u, y+k_n+v) \right| \quad , \quad \text{where}$$

$\{k_n | n = 0,..., p\}$, (u, v) is the motion vector and p=0,…, *N-1* which specifies the number of elements for producing the sum of errors for a partial SAD.

The resulting motion vector, $(\hat{u}, \hat{v})$, of a block is obtained by the following equation.

$$(\hat{u}, \hat{v}) \equiv \arg\min_{(u,v)} SAD_p(x, y; u, v)$$

For each block in other higher frequency subbands, the procedure is summarized as follows. (We use the LH subband as an example)

Step 5) Use Step 1 to Step 4 to obtain the adaptive index set in the highest resolution level LH subband.

Step 6) To obtain the adaptive index set for the lower resolution level, we down-sample the original set by discarding the index of the odd-sampled rows.

Step 7) Re-number the remaining index set to $n = 0,..., \frac{N}{2} - 1$ such that the row with larger expected absolute error will accumulate to the SAD_p as soon as possible.

Step 8) Repeat Step 4 by using the re-numbered index set to obtain the motion vectors in the next lower resolution level.

Step 9) Repeat Step 6 to Step 8 for remaining resolution levels.

3.4 Experimental results

In this experiment, we use eight test video sequences to evaluate the performance of the proposed algorithm. The information of the test sequences is shown in Table 3.1. Three levels of wavelet transformation are performed and the D4 kernel is used. The block sizes for the QCIF and CIF sequences are 2×2 and 4×4 respectively in the highest level and the block size will be doubled in each lower level. The search ranges for the QCIF and CIF sequences are ±8 and ±16 in the highest level respectively and it will be divided by two in each lower level.

Table 3.2 and Table 3.3 show the execution time and the average number of operations per block of not using the MRME scheme in wavelet domain respectively. All searching points are exhaustively searched by the searching algorithms. The full search for all subbands to obtain the motion vectors is very time consuming during encoding. It occupies more than 70% for the CIF video sequences and about 50% for the QCIF video sequences of encoding time. Therefore, there is a necessity to reduce the time required for motion estimation in wavelet domain. Actually, the pixel-based CPME-PDS can reduce the number of operations for a speed-up factor of 2.34 to 8.07 and 1.27 to 1.76 as compared with the FSA and PDS respectively for motion estimation as shown in Table 3.3. But its execution time is longer than that of the PDS as it suffers from random memory access problem in the CPU. Therefore, the row based CPME-

PDS can complement the deficiency of CPME-PDS by using a row of pixels as a unit for matching. During implementation, it can achieve a speed-up factor up to 1.12 as compared with the PDS. However, this scheme alone cannot fully make use of the wavelet property.

Table 3.1 The information of the test video sequence

Sequence	Frame Size	Total Frames
Trevor	176 × 144	120
Suzie	176 × 144	120
Salesman	176 × 144	300
Grandmother	176 × 144	300
Foreman	352 × 288	300
Coastguard	352 × 288	300
Akiyo	352 × 288	300
Vectra Colour	352 × 288	142

Table 3.2 Execution Time for motion estimation in searching all subbands by different search algorithms for video sequences in wavelet domain

Sequence	Execution time for motion estimation in wavelet domain (ms)							
	FSA	Speed-up factor	PDS	Speed-up factor	Pixel-based CPME-PDS	Speed-up factor	Row-based CPME-PDS	Speed-up factor
Trevor	15179	**1.00**	11278	**1.35**	11653	**1.30**	10584	**1.43**
Suzie	15357	**1.00**	13157	**1.17**	14898	**1.03**	12825	**1.19**
Salesman	37580	**1.00**	21992	**1.71**	22240	**1.69**	21149	**1.78**
Grandmother	36512	**1.00**	23859	**1.53**	26420	**1.38**	23792	**1.53**
Foreman	463636	**1.00**	288014	**1.61**	431311	**1.07**	276676	**1.68**
Coastguard	465801	**1.00**	254703	**1.83**	379112	**1.23**	248045	**1.88**
Akiyo	464424	**1.00**	144460	**3.21**	159772	**2.91**	129546	**3.59**
Vectra	220422	**1.00**	126515	**1.74**	180450	**1.22**	119345	**1.85**

Table 3.3 Average number of operations per block for motion estimation in searching all subbands by different search algorithms for video sequences in wavelet domain

Sequence	Average number of operations per block for motion estimation in wavelet domain							
	FSA	Speed-up factor	PDS	Speed-up factor	Pixel-based CPME-PDS	Speed-up factor	Row-based CPME-PDS	Speed-up factor
Trevor	97982	**1.00**	36873	**2.66**	26763	**3.66**	32175	**3.05**
Suzie	97982	**1.00**	50659	**1.93**	39957	**2.45**	47109	**2.08**
Salesman	97982	**1.00**	23493	**4.17**	17945	**5.46**	20685	**4.74**
Grandmother	97982	**1.00**	32735	**2.99**	24790	**3.95**	28957	**3.38**
Foreman	1533432	**1.00**	870525	**1.76**	654901	**2.34**	786480	**1.95**
Coastguard	1533432	**1.00**	741483	**2.07**	563793	**2.72**	673885	**2.28**
Akiyo	1533432	**1.00**	333742	**4.59**	190081	**8.07**	257495	**5.96**
Vectra	1521726	**1.00**	785792	**1.94**	565156	**2.69**	683723	**2.23**

The MRME (Multiresolution Motion Estimation) exploits the relationship between subbands by using the motion vectors found in the previous level as an initial estimate and refining in each level. In this experiment, the block size is still remained the same as the previous experiment. The search window of QCIF and CIF sequences are ±31 and ±63 respectively in the highest level and it is reduced by half in each lower level. The search window is set to be so large in the highest level because the MRME scheme makes use of the motion vectors obtained in the previous level to be an initial searching position, so the motion vectors in the top level must be accurate. Otherwise, the lower level subband will use the incorrect initial searching point and the inaccurate motion vector will be obtained.

Table 3.4 and Table 3.6 illustrate the execution time and the average number of operations per block of using the MRME scheme respectively in wavelet domain. According to Table 3.6, the row-based CPME-PDS reduces the number of operations in the encoder by rejecting the impossible candidates sooner especially for the refinement stage. Actually, the pixel-based CPME-PDS can reduce the operations in refinement but it also suffers from the random memory access problem in CPU. Due to the pipeline structure of CPU, the time used to access the pixels in a row is much shorter than that used to access the pixels in random location inside a block. Therefore, the pixel-based CPME-PDS can outperform other search algorithms theoretically. However, its execution time is longer than that of PDS due to the random memory access problem during implementation. The row-based CPME-PDS can achieve a speed-up factor of 1.01 to 1.97 and 1.01 to 1.04 as compared with FSA and conventional PDS respectively. Therefore, the row-based CPME-PDS will be used in the proposed algorithm.

Table 3.4 Execution Time for motion estimation using MRME scheme by different search algorithms for video sequences in wavelet domain

Execution Time for motion estimation in wavelet domain (ms)		Sequence							
		Trevor	Suzie	Salesman	Grand-mother	Foreman	Coast-guard	Akiyo	Vectra
FSA	Time	4580	4574	10857	11150	90922	91421	88471	43277
	Speed-up factor	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
PDS	Time	4421	4680	9768	10440	71847	73391	46570	34333
	Speed-up factor	1.04	0.98	1.11	1.07	1.27	1.25	1.90	1.26
Pixel-based CPME-PDS	Time	5053	5181	10955	11538	95095	100467	51870	45404
	Speed-up factor	0.91	0.88	0.99	0.97	0.96	0.91	1.71	0.95
Row-based CPME-PDS	Time	4750	4819	10598	10803	73000	75548	46664	34861
	Speed-up factor	0.96	0.95	1.02	1.03	1.25	1.21	1.90	1.24
Forward CPME-PDS	Time	8108	8535	18994	19665	121849	126264	83985	58623
	Speed-up factor	0.56	0.54	0.57	0.57	0.75	0.72	1.05	0.74
Backward CPME-PDS	Time	4289	4541	9935	10301	71454	73921	44902	33900
	Speed-up factor	1.07	1.01	1.09	1.08	1.27	1.24	1.97	1.28

Table 3.5 Average number of operations per block for motion estimation using MRME scheme by different search algorithms for video sequences in wavelet domain

Average number of operations per block for motion estimation in wavelet domain		Sequence							
		Trevor	Suzie	Salesman	Grand-mother	Foreman	Coast-guard	Akiyo	Vectra
FSA	Operations	13629	13681	13582	13552	228807	228990	224318	228369
	Speed-up factor	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
PDS	Operations	6455	8330	4341	5637	138186	142598	49089	137684
	Speed-up factor	**2.11**	**1.64**	**3.12**	**2.40**	**1.66**	**1.67**	**4.57**	**1.66**
Pixel-based CPME-PDS	Operations	5317	6970	3936	4921	105083	115335	33123	104614
	Speed-up factor	**2.56**	**1.96**	**3.44**	**2.75**	**2.18**	**1.99**	**6.77**	**2.18**
Row-based CPME-PDS	Operations	5835	7643	4136	5203	122452	129773	39217	120511
	Speed-up factor	**2.34**	**1.79**	**3.27**	**2.60**	**1.87**	**1.76**	**5.72**	**1.90**
Forward CPME-PDS	Operations	6296	8151	4352	5569	130307	137698	43393	129719
	Speed-up factor	**2.16**	**1.68**	**3.11**	**2.43**	**1.76**	**1.66**	**5.17**	**1.76**
Backward CPME-PDS	Operations	5885	7759	4100	5211	125679	132880	40171	123877
	Speed-up factor	**2.32**	**1.76**	**3.30**	**2.60**	**1.82**	**1.72**	**5.58**	**1.84**

Besides the above proposed scheme (entitled as Backward CPME-PDS in this section), we propose one more variation to compare the efficiency of the proposed method. It is the Forward CPEM-PDS which carries out the counting sort in the subbands of the highest level only and the results are up-sampled in the remaining subbands of the lower level. For the Backward CPME-PDS, i.e. the proposed scheme, the counting sort will be executed in the LL subband and the subbands at the lowest level only. Then, the sorting results obtained in the subbands of the lowest level are up-sampled to predict the adaptive index set in the higher levels.

The Forward CPME-PDS carries out counting sort in the subbands of the highest level and the results are re-numbered to predict the error distribution of the subbands in the remaining lower levels. However, the prediction is inaccurate leading to

a decrease in the speed of motion estimation. In our experimental work, the block size used in the 3rd level was 4×4, 8×8 in 2nd level and 16×16 in 1st level for the CIF format video sequence. After performing counting sort of the subbands in the 3rd level, the result was applied to the corresponding subbands in the 2nd level directly. The first two rows were grouped into one unit and the third and fourth rows were grouped into another unit, etc. Some information in the subbands at the 3rd level is invalid in the corresponding subbands at the 2nd level resulting in some inaccurate predictions. Hence, the impossible candidate is not rejected as soon as expected. As a result, the efficiency of the Forward CPME-PDS is degraded as a comparison with the Row-based CPME-PDS. According to Table 3.4 and Table 3.6, the execution time and the average number of operations per block of the Forward CPME-PDS are both greater than that of the Row-based CPME-PDS. During Discrete Wavelet Transformation, the higher resolution level subbands are downsampled to obtain the subbands in the next lower resolution level. Hence, the information in the subbands in higher resolution level can be used to predict that of the lower one due to the hierarchical architecture in wavelet pyramid. As a result, the Backward CPME-PDS outperforms the Row-based CPME-PDS in terms of execution time by a speed-up factor of 1.02 to 1.11 due to an accurate prediction of the adaptive index set by exploiting the hierarchical clustering property in wavelet domain. The impossible candidates can be rejected earlier.

As mentioned in chapter 2, the Successive Elimination Algorithm (SEA) can successively reduce the search positions inside the search area and the motion vectors obtained are the same as that of the FSA. Therefore, we apply the SEA into the Backward CPME-PDS, entitled as “SEA & Backward CPME-PDS”, in order to further enhance the speed of the motion estimation. Table 3.7 and Table 3.8 show the execution time and the average number of operations per block of the proposed algorithm

respectively. For the average number of operations per block, the proposed algorithm (SEA & Backward CPME-PDS) can achieve a speed-up factor of 1.86 to 10.30, 1.05 to 1.80 and 1.01 to 1.08 as compared with FSA, Backward CPME-PDS and SEA&PDS respectively.

Table 3.6 Execution Time for motion estimation using MRME scheme by different search algorithms for video sequences in wavelet domain

Execution Time for motion estimation in wavelet domain (ms)		Sequence							
		Trevor	Suzie	Salesman	Grand-mother	Foreman	Coast-guard	Akiyo	Vectra
FSA	Time	4580	4574	10857	11150	90922	91421	88471	43277
	Speed-up factor	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
PDS	Time	4421	4680	9768	10440	71847	73391	46570	34333
	Speed-up factor	1.04	0.98	1.11	1.07	1.27	1.25	1.90	1.26
Backward CPME-PDS	Time	4289	4541	9935	10301	71454	73921	44902	33900
	Speed-up factor	1.07	1.01	1.09	1.08	1.27	1.24	1.97	1.28
SEA & FSA	Time	4372	4639	10196	10530	85964	84993	47276	39291
	Speed-up factor	1.05	0.99	1.06	1.06	1.06	1.08	1.87	1.10
SEA & PDS	Time	4307	4665	9850	10218	75584	78811	42267	36065
	Speed-up factor	1.06	0.98	1.10	1.09	1.20	1.16	2.09	1.20
SEA & Backward CPME-PDS	Time	4315	4648	9877	10266	72463	75957	41004	34630
	Speed-up factor	1.06	0.98	1.10	1.09	1.25	1.20	2.16	1.25

Table 3.7 Average number of operations per block for motion estimation using MRME scheme by different search algorithms for video sequences in wavelet domain

Average number of operations per block for motion estimation in wavelet domain		Sequence							
		Trevor	Suzie	Salesman	Grand-mother	Foreman	Coast-guard	Akiyo	Vectra
FSA	Operations	13629	13681	13528	13552	228807	228990	224318	228369
	Speed-up factor	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
PDS	Operations	6455	8330	4341	5637	138186	142598	49089	137684
	Speed-up factor	**2.11**	**1.64**	**3.12**	**2.40**	**1.66**	**1.61**	**4.57**	**1.66**
Backward CPME-PDS	Operations	5885	7759	4100	5211	125679	132880	40171	123877
	Speed-up factor	**2.32**	**1.76**	**3.30**	**2.60**	**1.82**	**1.72**	**5.58**	**1.84**
SEA & FSA	Operations	6830	8981	4789	6023	172022	178729	46625	170912
	Speed-up factor	**2.00**	**1.52**	**2.82**	**2.25**	**1.33**	**1.28**	**4.81**	**1.34**
SEA & PDS	Operations	4608	6838	2555	3830	121542	129372	23460	120857
	Speed-up factor	**2.96**	**2.00**	**5.29**	**3.54**	**1.88**	**1.77**	**9.56**	**1.89**
SEA & Backward CPME-PDS	Operations	4558	6683	2721	3888	113595	123144	21784	112591
	Speed-up factor	**2.99**	**2.05**	**4.97**	**3.49**	**2.01**	**1.86**	**10.30**	**2.03**

Table 3.9 shows the average number of search positions per block for the exhaustively searching algorithm, MRME scheme and SEA applied in the MRME scheme. When SEA is applied in the MRME scheme, the speed-up factor can be from 4.64 to 26.69 and 1.75 to 10.31 as compared with the exhaustively search algorithm and MRME scheme respectively. As the SEA can remove the searching position in the search range, so the number of operations can be reduced. For execution time, the performance of the SEA & Backward CPME-PDS is degraded as compared with the Backward CPME-PDS for some video sequences. It is because the number of operations is the summation of the number of additions or subtraction and the number of comparisons. When the SEA is used, the number of comparisons is increased but the number of additions or subtractions is reduced significantly. However, the time used to

perform comparison is longer than that of the additions or subtractions in CPU. Therefore, the execution time for the SEA & Backward CPME-PDS algorithm is increased as compared with that of the Backward CPME-PDS even though the number of operations is reduced. The "Akiyo" sequence contains very slow motion, so the SEA can quickly reject most searching positions in the search window. Therefore, when the SEA is combined with the Backward CPME-PDS, the execution time and the average number of operations per block are both reduced for a speed-up factor of 1.10 and 1.84 respectively as compared with the Backward CPME-PDS for "Akiyo" sequence. The SEA & Backward CPME-PDS algorithm can enhance the speed of motion estimation for the very slow video sequence.

Table 3.8 Average number of search points per block for motion estimation in wavelet domain

Average number of search points per block for motion estimation in wavelet domain		Sequence							
		Trevor	**Suzie**	**Salesman**	**Grand-mother**	**Foreman**	**Coast-guard**	**Akiyo**	**Vectra**
FSA	**Search pts.**	967	967	967	967	3763	3763	3763	3763
	Speed-up factor	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
MRME	**Search pts.**	366	366	364	364	1466	1470	1454	1460
	Speed-up factor	**2.64**	**2.64**	**2.66**	**2.66**	**2.57**	**2.56**	**2.59**	**2.58**
SEA & MRME	**Search pts.**	88	120	53	65	742	838	141	811
	Speed-up factor	**10.99**	**8.06**	**18.25**	**14.88**	**5.07**	**4.49**	**26.69**	**4.64**

3.5 Conclusion

Wavelet-based motion estimation algorithm is proposed in this chapter. The pixel cluster property is available in both spatial and wavelet domains. Besides, this clustering property is appeared in the hierarchical nature of the wavelet pyramid. By applying the hierarchical property of the wavelet domain into the row-based CPME-PDS, the impossible candidate blocks can be rejected as early as possible. Only the LL subband and the subbands in the lowest level have to carry out counting sort to find their error distributions. The results can then be down-sampled and re-numbered in the higher level subbands in the hierarchical pyramid. Experimental results show that the proposed scheme has a speed-up factor of 1.09 to 2.16 and 1.86 to 10.30 in the execution time and the average number of operations per block respectively for motion estimation comparing to FSA in MRME scheme. It performs well for the slow motion video sequences. Hence, it is suitable for video conferencing and video surveillance. Due to the scalability nature of the DWT, the proposed method can be applied to multi-resolution applications such as DTV, HDTV and mobile phone applications.

Chapter 4

Motion estimation algorithm in the wavelet domain of the 3D wavelet video coder

4.1 Introduction

The Discrete Wavelet Transform (DWT) has received much attention recently due to its superior performance by comparing to the conventional block-based hybrid video coding such as Discrete Cosine Transform (DCT). It is well-known that the DCT produces the "blocking effect" in the low bit-rate applications. However, DWT is free from blocking artifacts by distributing the errors over the whole frame due to its nature of global decomposition. Besides, according to its multi-resolution nature in the wavelet pyramid, it can represent an image or video sequence flexibly and adapt to different bit-rates across the networks with different traffic situations. Recently, the scalable video coder based on the motion compensation spatiotemporal (*t+2D*) scheme becomes much more popular. It can achieve superior compression performance as compared with the state-of-the-art DCT based hybrid video-coding scheme. Its excellent compression performance comes from the efficient energy concentration of the low frequency frame by applying motion compensation along the motion trajectories of the video sequence. Besides, the separable three-dimensional (3D) wavelet transform can realize the temporal and spatial scalabilities in the subband structure.

Low encoding delay is a necessity in the video conferencing and video surveillance applications. For example, in the video surveillance application, it is possible to take actions to follow up the emergency events if the video sequence can be

encoded and transmitted in a short period of time. On the other hand, the multiresolution nature of the DWT is suitable for multi-resolution applications such as DTV / HDTV. Besides, when a video sequence is transmitted to the low-end display units such as mobile phone and PDA, the required resolution should be achieved before displaying.

In this chapter, we exploit the spatial and temporal correlations of the neighbouring motion vectors and the successive video frames respectively to find out the initial estimated motion vector. Then, the refinement process is performed in the reduced search window. The resultant motion vector is obtained by adding the initial estimated motion vector and the small refinement motion vector. Since the size of the search window is decreased, the computational complexity of motion estimation can also be reduced. From the experimental results, the proposed algorithm can achieve a speed-up factor of 3 to 5 times as compared to that of the conventional approach [52]. Besides, the reconstructed quality of the proposed algorithm is comparable to the FSA.
The organization of this chapter is shown as follows. Details of the proposed algorithm is introduced in Section 4.2. Some experimental results are discussed in Section 4.3 and a brief conclusion is drawn in Section 4.4.

4.2 Proposed wavelet-domain motion estimation algorithm in the 3D wavelet video coder

Figure 4.1 shows the architecture of the 3D wavelet video codec using Haar kernel during 1D temporal wavelet transformation. Before performing temporal decomposition, motion estimation is carried out so that the ghosting artifacts occurred in the low frequency frame can be reduced and most energy concentrates on the low

frequency frame which can improve the compression efficiency [52]. The temporal decomposition is then performed on the motion compensated frames. Then, the 2D spatial wavelet transform is applied on the temporally filtered frames. Finally, the 3D SPIHT [109] or MC-EZBC [110] are used to convert the wavelet coefficients into bitstream. Traditionally, the motion vectors in each temporal level are obtained independently. As shown in Figure 4.1, the first low frequency frame in temporal level 1 is the average frame of frames 0 and 1 in temporal level 0. Therefore, there are large correlations existing between them. If these temporal correlations can be exploited, the speed of motion estimation will be enhanced.

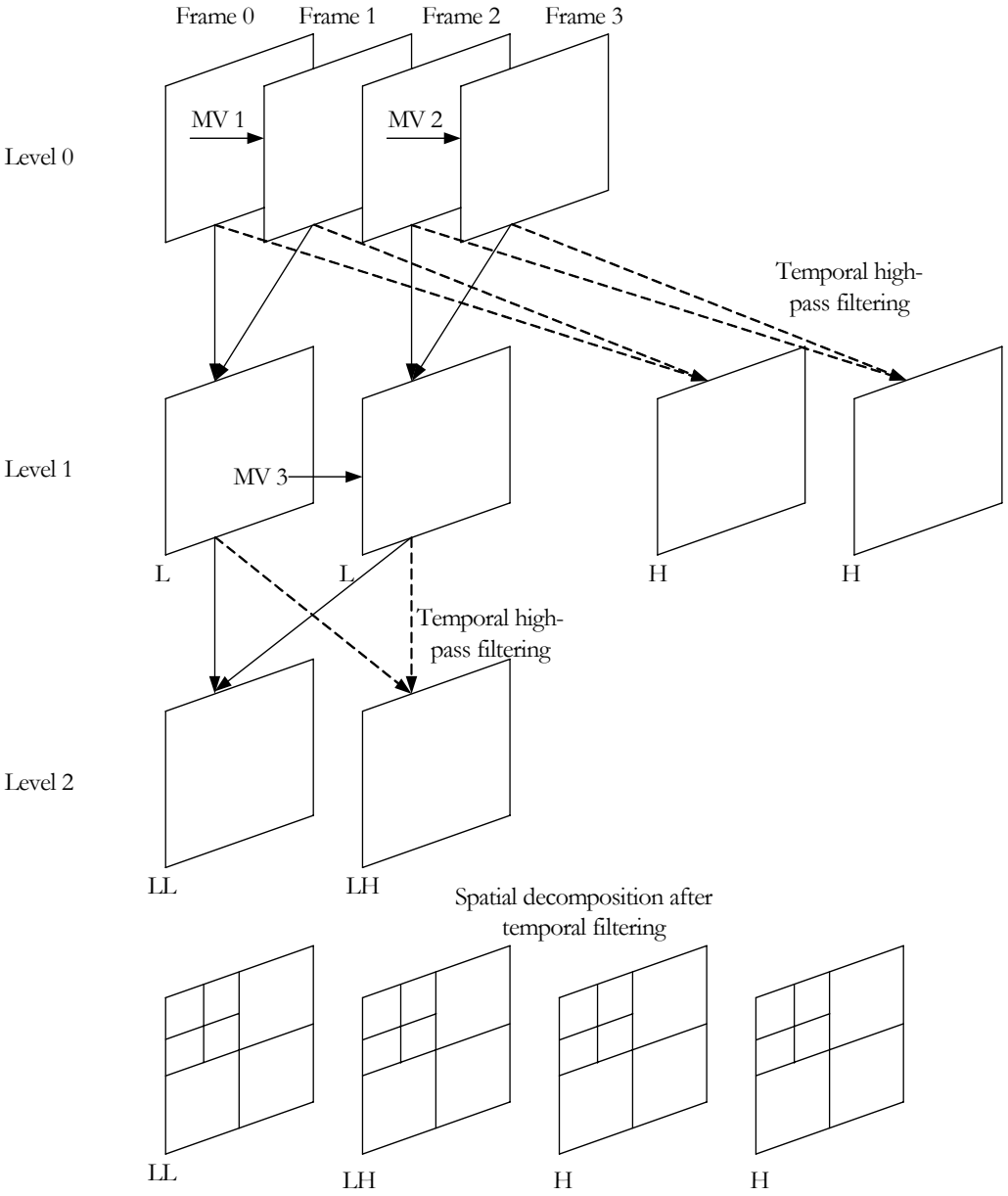


Figure 4.1 Architecture of the 3D wavelet video coder

4.2.1 Cross-level prediction of motion vector

For the sake of simplicity, the Haar kernel is used as an example to illustrate the idea of the proposed algorithm. In Figure 4.1, four frames are formed as a Group Of Frames (GOF) and two levels of the temporal decomposition is performed. Firstly, the forward motion vectors, MV1 and MV2, are obtained by motion estimation performing on the frames 0 and 1 and frames 2 and 3 respectively where frames 0 and 2 are the

reference frames. Then, the temporal wavelet transformation is applied on the motion compensated frames and the low and high frequency frames are calculated by the lifting equations. Before carrying out temporal decomposition in the temporal level 1, motion estimation is performed in order to find the forward motion vector, MV3, in temporal level 1. At this stage, the temporal correlation between different temporal levels can be exploited. As depicted in Figure 4.2, the average of the MV1 and MV2 is used as the initial estimated position of MV3 (think arrow). Then, the refinement process is performed in the reduced search window (dotted square). This is to find the refined motion vector as indicated by the dotted arrow in the Figure 4.3. This resultant motion vector, MV3, is obtained by adding the initial estimated motion vector to the refinement motion vector. As the size of the search window is reduced, so the computational complexity can be decreased. Similarly, the backward motion vector is calculated in the same way. This concept can be extended to the kernel with long filter length such as the Bi-orthogonal 5/3 kernel.

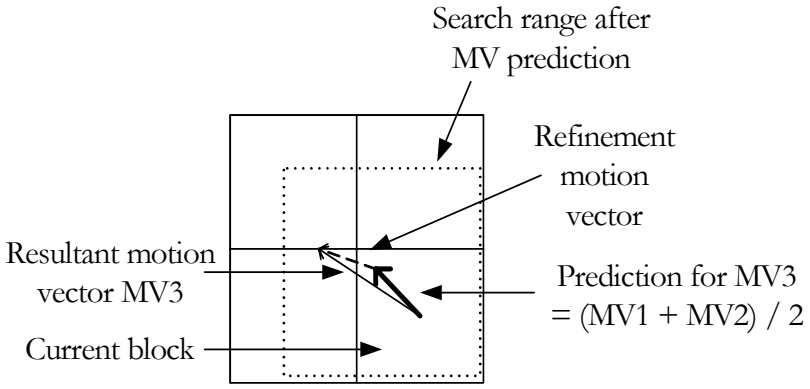


Figure 4.2 Cross-level motion vector prediction

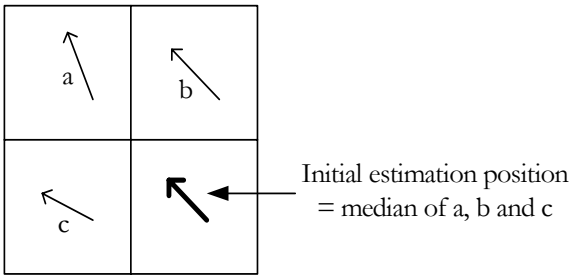


Figure 4.3 Median prediction

4.2.2 Median prediction of motion vector

From our experimental work, the time spent on motion estimation in the temporal level 0, i.e. the original video sequence, is around half of the total time for motion estimation. Thus, the time consumed in the motion estimation of temporal level 0 is the most consuming part in the whole motion estimation procedure. There exists a large correlation between neighbouring motion vectors, so differential coding is used to encode the motion vectors. If such correlation can be exploited, the motion estimation operation can be further reduced. The median of the motion vectors located in the top-left, top and left of the current block is considered as the initial estimated position as depicted in Figure 4.3. Similar to the algorithm mentioned in Section 4.1, the refinement process is performed in the reduced search area. The resultant motion vector of the current block is the sum of the initial estimated motion vector and the small refined motion vector. Since the size of the search area is decreased, the number of operations for motion estimation can be reduced. As a result, the speed of the motion estimation can be further improved.

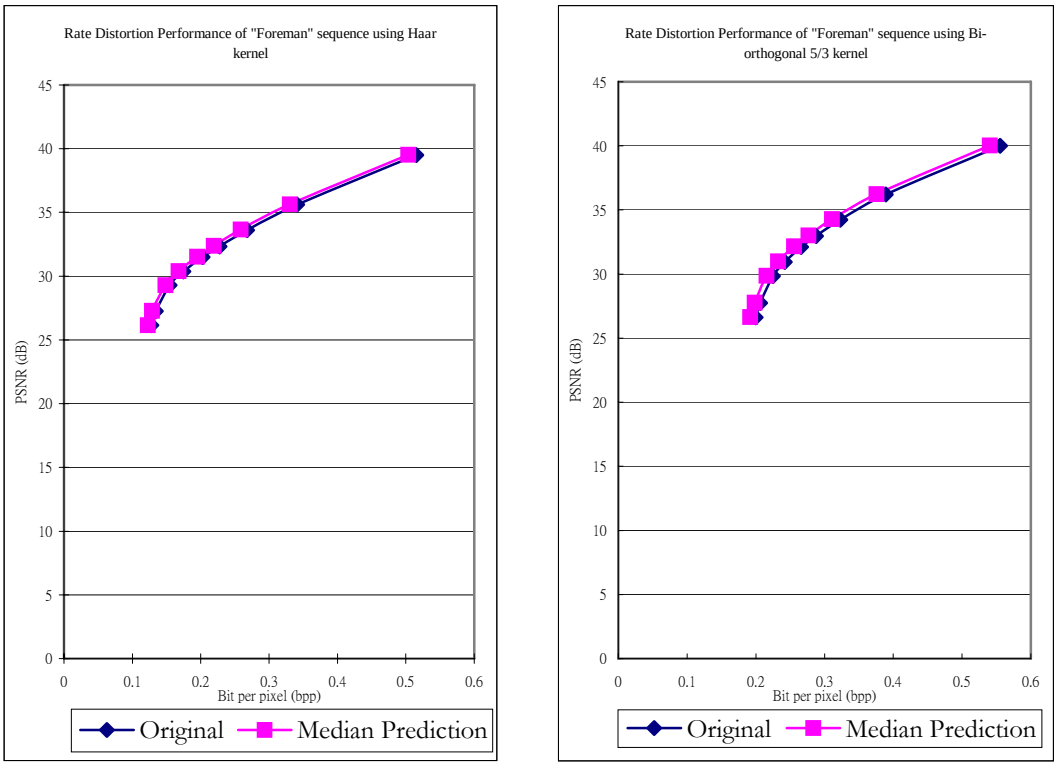
4.3 Experimental Results

A substantial amount of experimental work has been done on our approach. Consider the results for the block-based motion model with the block size of 16×16 and integer pixel accuracy in motion estimation. The search window in the refinement process is reduced by half in each temporal level. For example, if the search window in the temporal level 0 is ±8, the size of the search window in the temporal level 1 will be ±4, etc. In our experiments, as the median prediction was used in temporal level 0, so a reduced search window with the size of ±4 is used. The search windows in temporal levels 1 and 2 are ±4 and ±2 respectively. The motion vectors are encoded by Variable Length Coding (VLC). Three levels of temporal and spatial decompositions are performed and the Bi-orthogonal 9/7 kernel is used during the spatial decomposition. After temporal and spatial decompositions, the spatio-temporal wavelet coefficients were encoded by Embedded Zerotree Wavelet (EZW) [57] coding and the Huffman coding to convert into bitstream.

Eight video sequences were tested. They are the “Trevor”, “Suzie”, “Salesman” and “Grandmother” with QCIF format (120 frames), and the “Foreman”, “Akiyo” and “Coastguard” with CIF format and the “Stefan” with SIF format (296 frames) at 30 fps.

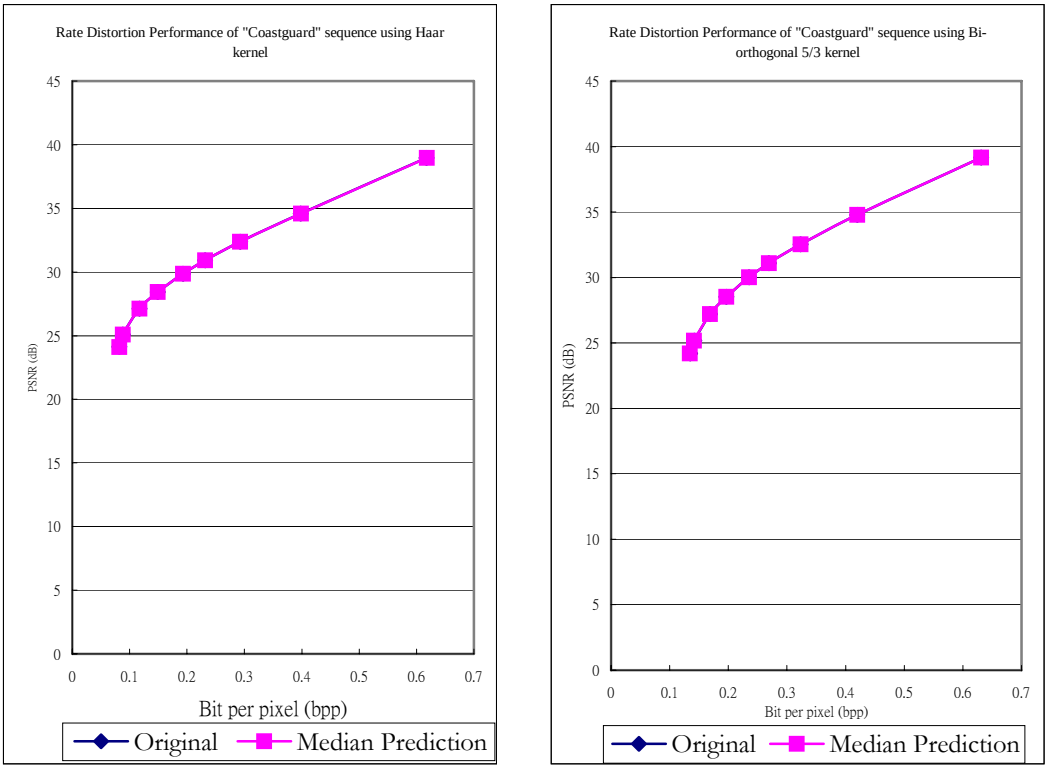
Table 4.1 and Table 4.2 illustrate the execution time used in motion estimation by using the cross-level motion vector prediction and row-based CPME-PDS to find the motion vectors in different temporal levels for the Haar and Bi-orthogonal 5/3 kernels respectively. The cross-level motion vector prediction makes use of the consecutive motion vectors in the previous temporal level for initial searching points and performs the refinement in a reduced search window in order to reduce the number of operations used in searching. This scheme can achieve the speed up factor of 2.83 to 5.80 and 3.94 to 7.25 in temporal levels 1 and 2 respectively for Haar kernel and 2.76 to 5.69 and 3.59

to 7.42 in temporal levels 1 and 2 respectively for Bi-orthogonal 5/3 kernel. The improvement in speed up for the total execution time for motion estimation in MCTF is from 2.05 to 4.18 and 1.99 to 4.13 for Haar and Bi-orthogonal 5/3 kernels respectively as compared with the FSA. During encoding, the MCTF can attain the speed-up factor of 1.89 to 3.33 and 1.86 to 3.40 for Haar and Bi-orthogonal 5/3 kernels respectively in terms of processing time. Figure 4.4, Figure 4.5 and Figure 4.6 depicts the rate-distortion performance of the cross-level motion vector prediction scheme in finding the motion vectors at different temporal levels for the "Foreman", "Coastguard" and "Stefan" sequences respectively. As there exists large correlation between temporal levels, this prediction scheme is accurate to find the initial searching position. Therefore, the reconstructed quality is comparable to that of the FSA.



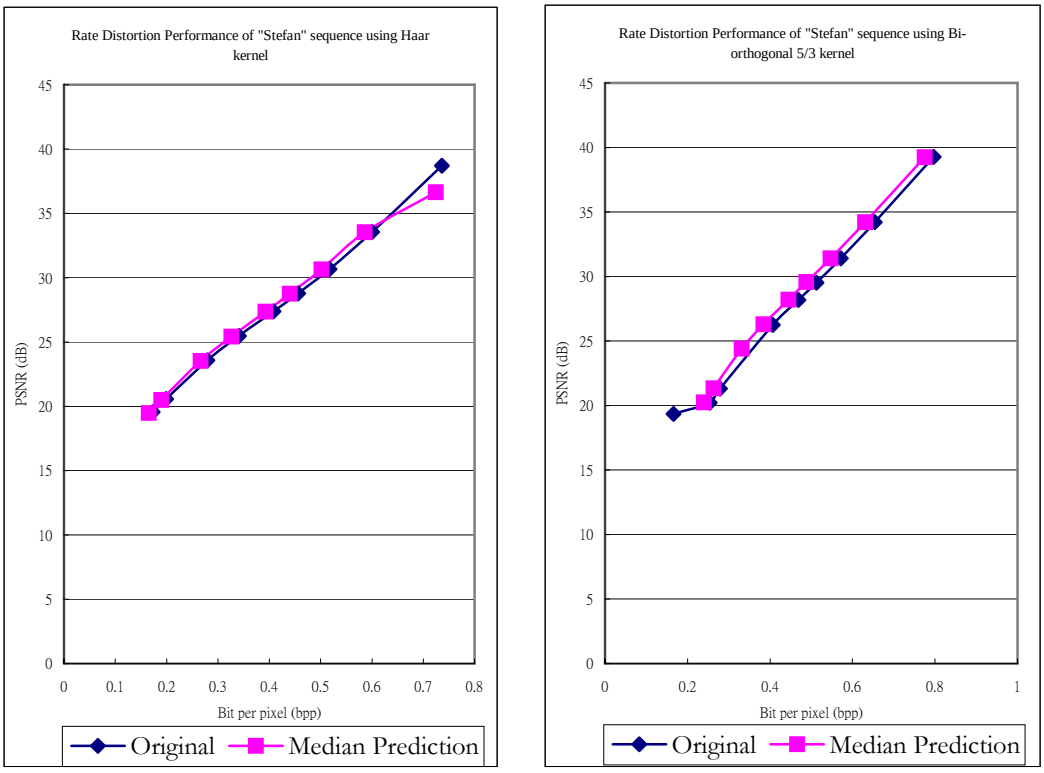
(a) **(b)**

Figure 4.4 Rate distortion performance of "Foreman" sequence for median prediction using (a) Haar kernel and (b) Bi-orthogonal 5/3 kernel during temporal decomposition



(a) **(b)**

Figure 4.5 Rate distortion performance of "Coastguard" sequence for median prediction using (a) Haar kernel and (b) Bi-orthogonal 5/3 kernel during temporal decomposition



(a) **(b)**

Figure 4.6 Rate distortion performance of "Stefan" sequence for median prediction using (a) Haar kernel and (b) Bi-orthogonal 5/3 kernel during temporal decomposition

Table 4.1 Total execution time used in motion estimation (ms) for cross-level motion vector prediction using Haar kernel during temporal decomposition

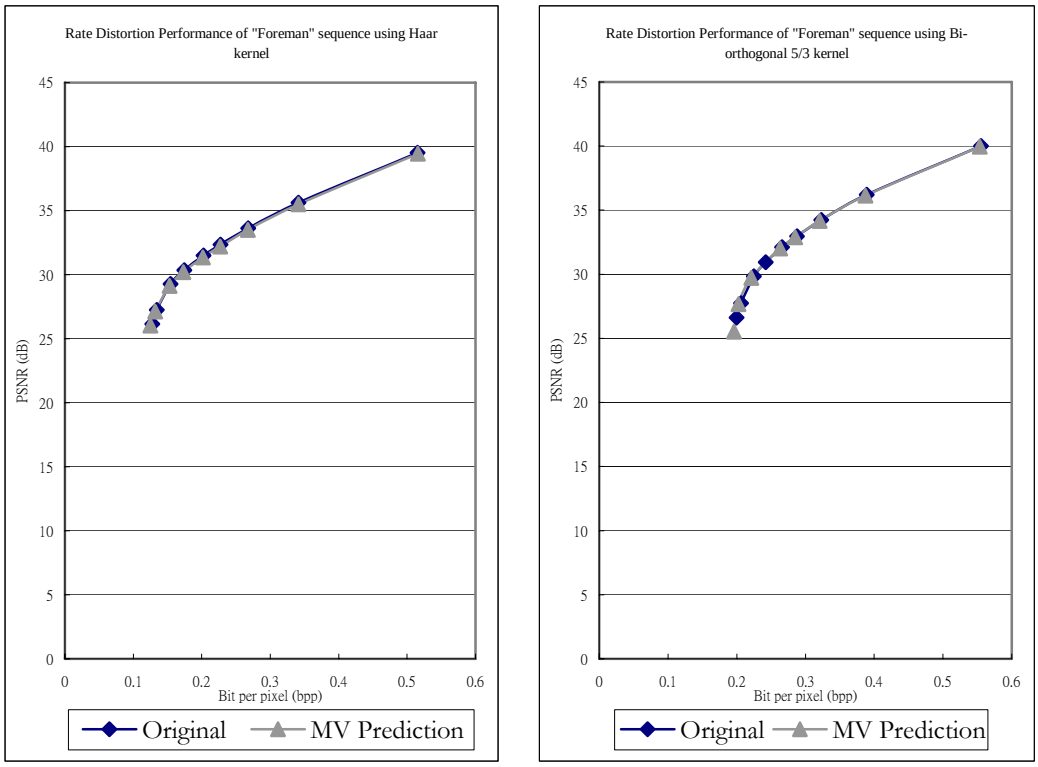
Video sequence	Execution Time used for motion estimation (ms)									Execution Time used for temporal decomposition (ms)		
	level 1 (no med. + no mv pred. + FSA)	level 1 (mv pred. + CPME -PDS)	speed-up factor	level 2 (no med. + no mv pred. + FSA)	Level 2 (mv pred. + CPME -PDS)	speed-up factor	total (no med. + no mv pred. + FSA)	total (mv pred. + CPME -PDS)	speed-up factor	no med. + no mv pred. + FSA	mv pred. + CPME -PDS	speed-up factor
Trevor	814	211	**3.86**	421	91	**4.63**	2751	982	**2.80**	2923	1222	**2.39**
Suzie	829	237	**3.50**	372	93	**4.00**	2715	1092	**2.49**	2982	1327	**2.25**
Salesman	1889	434	**4.35**	921	179	**5.15**	6623	1959	**3.38**	7171	2538	**2.83**
Grandmother	1873	508	**3.69**	1031	191	**5.40**	6651	2428	**2.74**	7188	3001	**2.40**
Foreman	9170	2760	**3.32**	4545	996	**4.56**	31950	13491	**2.37**	34469	15983	**2.16**
Akiyo	8736	1505	**5.80**	4374	603	**7.25**	30462	7296	**4.18**	32717	9811	**3.33**
Coastguard	8939	2681	**3.33**	4470	1004	**4.45**	31426	12543	**2.51**	33420	15051	**2.22**
Stefan	7387	2607	**2.83**	3755	954	**3.94**	25975	12644	**2.05**	27769	14723	**1.89**

Table 4.2 Total execution time used in motion estimation (ms) for cross-level motion vector prediction using Bi-orthogonal 5/3 kernel during temporal decomposition

Video sequence	Execution Time used for motion estimation (ms)									Execution Time used for temporal decomposition (ms)		
	level 1 (no med. + no mv pred. + FSA)	level 1 (mv pred. + CPME -PDS)	speed-up factor	level 2 (no med. + no mv pred. + FSA)	Level 2 (mv pred. + CPME -PDS)	speed-up factor	total (no med. + no mv pred. + FSA)	total (mv pred. + CPME -PDS)	speed-up factor	no med. + no mv pred. + FSA	mv pred. + CPME -PDS	speed-up factor
Trevor	1203	367	**3.28**	406	107	**3.79**	4345	1714	**2.54**	4705	2042	**2.30**
Suzie	1201	349	**3.44**	345	96	**3.59**	4312	1772	**2.43**	4687	2120	**2.21**
Salesman	2891	664	**4.35**	788	184	**4.28**	10155	3165	**3.21**	11029	3988	**2.77**
Grandmother	2845	776	**3.67**	890	203	**4.38**	10533	4008	**2.63**	11221	4812	**2.33**
Foreman	13750	4205	**3.27**	4577	1043	**4.39**	50201	22348	**2.25**	53690	25928	**2.07**
Akiyo	13314	2340	**5.69**	4461	601	**7.42**	48809	11821	**4.13**	52242	15348	**3.40**
Coastguard	13536	4122	**3.28**	4538	1058	**4.29**	50126	20477	**2.45**	53574	24001	**2.23**
Stefan	11262	4075	**2.76**	3778	983	**3.84**	41385	20801	**1.99**	44297	23761	**1.86**

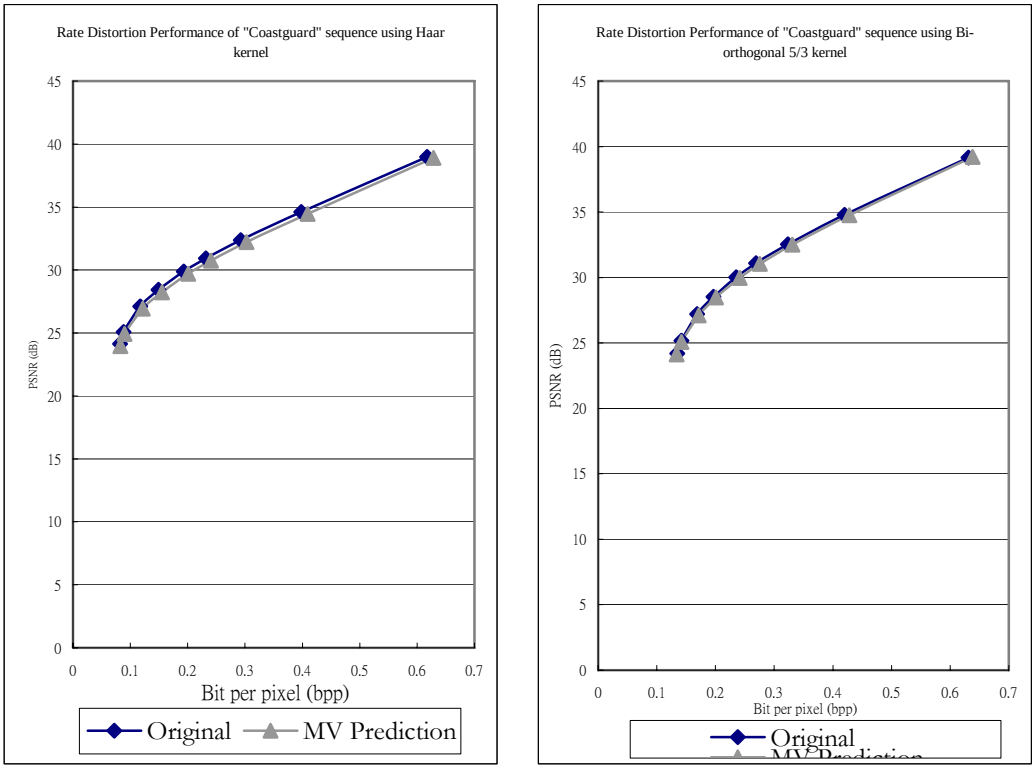
The execution time for motion estimation using the row-based CPME-PDS and median prediction scheme using Haar and Bi-orthogonal 5/3 kernels are shown in Table 4.3 and Table 4.4 respectively. This scheme can achieve a speed-up factor of 2.94 to 4.50 and 3.31 to 5.55 for Haar and Bi-orthogonal 5/3 kernels respectively in temporal level 0 as compared with FSA. As the Bi-orthogonal 5/3 kernel involves bi-directional prediction in both predict and update steps, the number of motion vector fields is more than that of the Haar kernel. Therefore, the speed-up factor is higher as compared with

Haar kernel. The PSNR values of reconstructed sequences at different bitrates are shown in Figure 4.7, Figure 4.8 and Figure 4.9 for “Foreman”, “Coastguard” and “Stefan” respectively. As the correlation of the motion vectors among neighbouring blocks is very high, so the median prediction scheme can provide an accurate prediction of initial searching point. As a result, this scheme can attain similar rate-distortion performance as compared with the FSA.



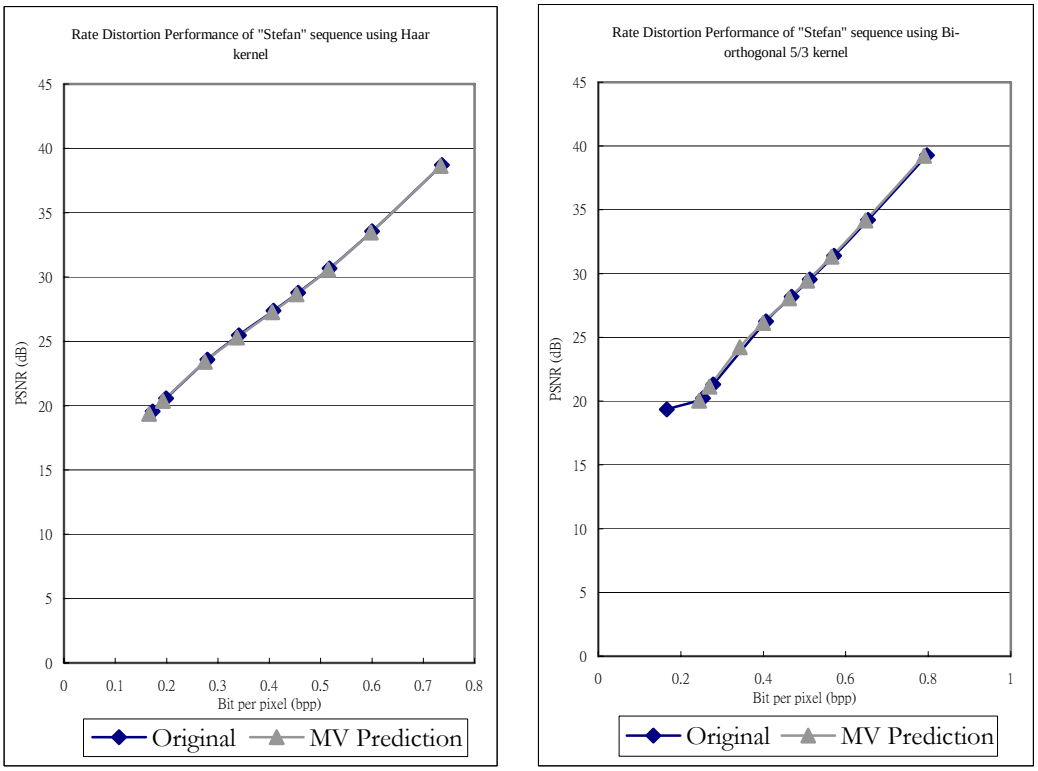
(a) **(b)**

Figure 4.7 Rate distortion performance of “Foreman” sequence for cross-level motion vector prediction using (a) Haar kernel and (b) Bi-orthogonal 5/3 kernel during temporal decomposition



(a) **(b)**

Figure 4.8 Rate distortion performance of "Coastguard" sequence for cross-level motion vector prediction using (a) Haar kernel and (b) Bi-orthogonal 5/3 kernel during temporal decomposition



(a) **(b)**

Figure 4.9 Rate distortion performance of "Stefan" sequence for cross-level motion vector prediction using (a) Haar kernel and (b) Bi-orthogonal 5/3 kernel during temporal decomposition

Table 4.3 Total execution time used in motion estimation (ms) for median prediction using Haar kernel during temporal decomposition

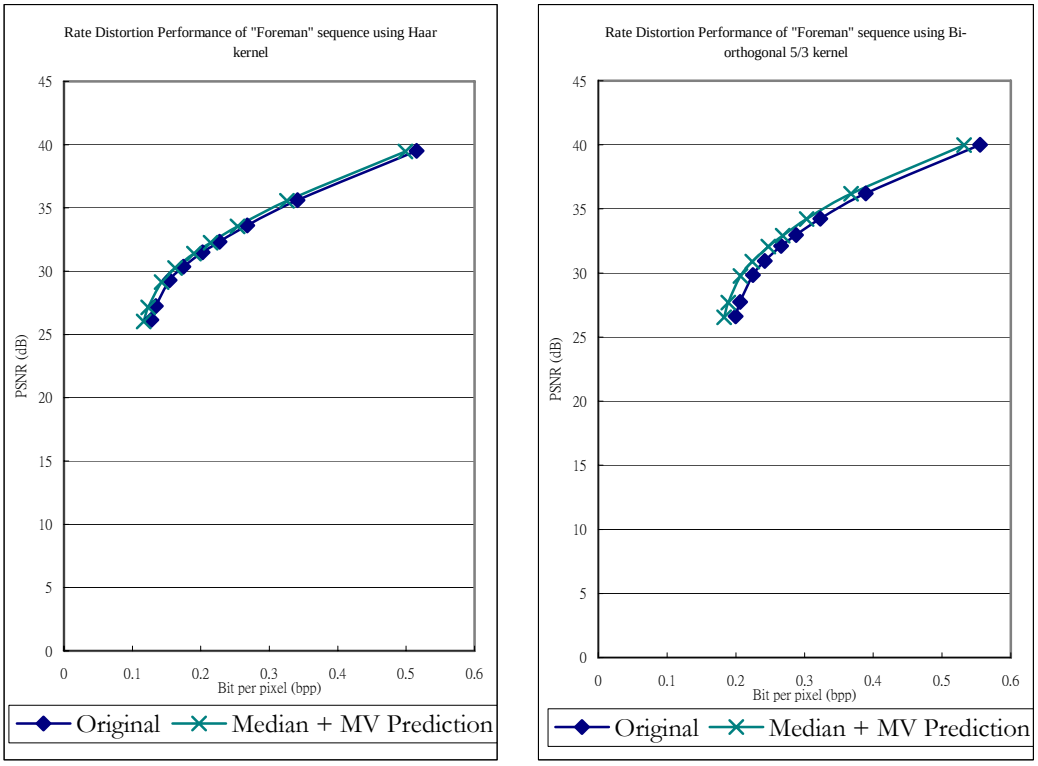
Video sequence	Execution Time used for motion estimation (ms)						Execution Time used for temporal decomposition (ms)		
	level 0 (no med. + no mv pred. + FSA)	level 0 (med. + CPME -PDS)	speed-up factor	total (no med. + no mv pred. + FSA)	total (med. + CPME -PDS)	speed-up factor	no med. + no mv pred. + FSA	med. + CPME -PDS	speed-up factor
Trevor	1516	455	**3.33**	2751	1080	**2.55**	2923	1468	**1.99**
Suzie	1514	517	**2.93**	2715	1249	**2.17**	2982	1471	**2.03**
Salesman	3813	1052	**3.62**	6623	2285	**2.90**	7171	2847	**2.52**
Grandmother	3747	1171	**3.20**	6651	2701	**2.46**	7188	3188	**2.25**
Foreman	18235	5716	**3.19**	31950	14900	**2.14**	34469	17484	**1.97**
Akiyo	17352	3855	**4.50**	30462	8432	**3.61**	32717	10674	**3.07**
Coastguard	18017	5223	**3.45**	31426	14458	**2.17**	33420	17188	**1.94**
Stefan	14833	5046	**2.94**	25975	14125	**1.84**	27769	16199	**1.71**

Table 4.4 Total execution time used in motion estimation (ms) for median prediction using Bi-orthogonal 5/3 kernel during temporal decomposition

Video sequence	Execution Time used for motion estimation (ms)						Execution Time used for temporal decomposition (ms)		
	level 0 (no med. + no mv pred. + FSA)	level 0 (med. + CPME -PDS)	speed-up factor	total (no med. + no mv pred. + FSA)	total (med. + CPME -PDS)	speed-up factor	no med. + no mv pred. + FSA	med. + CPME -PDS	speed-up factor
Trevor	2736	763	**3.59**	4345	1671	**2.60**	4705	2063	**2.28**
Suzie	2766	731	**3.78**	4312	1478	**2.92**	4687	1858	**2.52**
Salesman	6476	1531	**4.23**	10155	2997	**3.39**	11029	3857	**2.86**
Grandmother	6798	1799	**3.78**	10533	3658	**2.88**	11221	4532	**2.48**
Foreman	31874	8594	**3.71**	50201	20157	**2.49**	53690	24281	**2.21**
Akiyo	31034	5593	**5.55**	48809	11076	**4.41**	52242	14488	**3.61**
Coastguard	32052	8572	**3.74**	50126	20482	**2.45**	53574	24160	**2.22**
Stefan	26345	7958	**3.31**	41385	19455	**2.13**	44297	22512	**1.97**

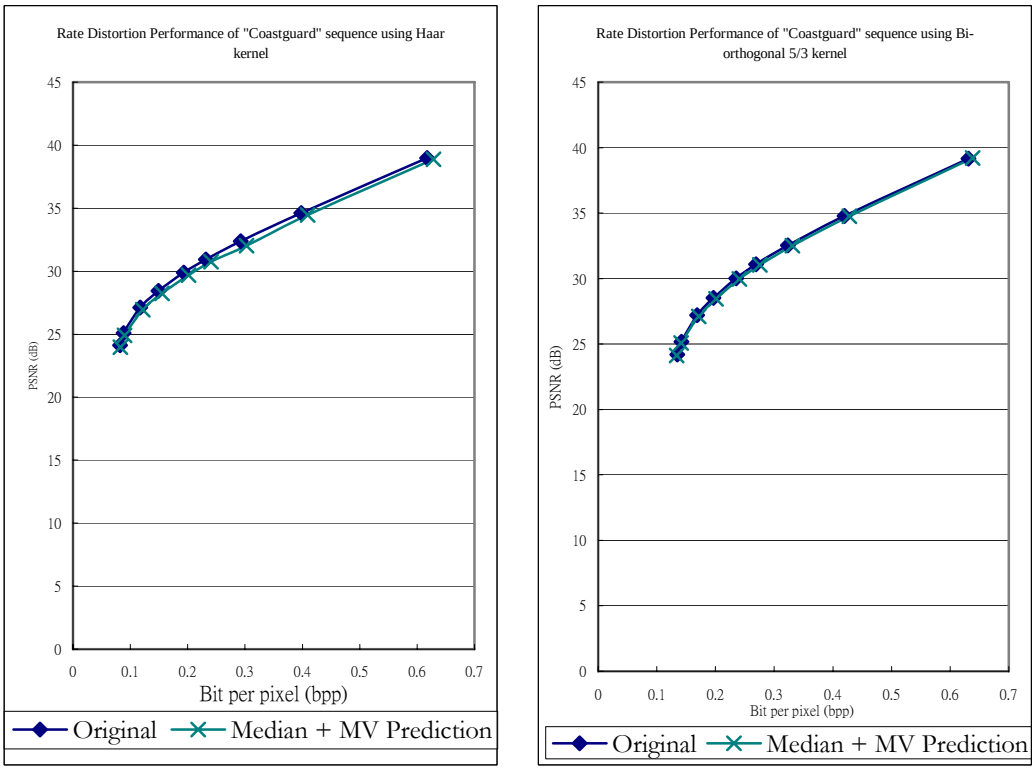
Table 4.5 and Table 4.6 show the results using the row-based CPME-PDS, median prediction and cross-level motion vector prediction in obtaining the motion vectors of different temporal levels for the Haar and Bi-orthogonal 5/3 kernels in temporal decomposition respectively. At temporal levels 0 and 1, the proposed algorithm can achieve a speed-up factor from 3 to 5 times as compared with the FSA without using median and cross-level motion vector predictions. Since the search range at these two temporal levels is the same, i.e. ±4, the proposed algorithm can obtain

similar speed-up factor in these two temporal levels. At temporal level 2, the speed-up factor can be further increased. As the search window is ±8 at all temporal levels in the original algorithm and the reduced search window, i.e. ±2, is used in the proposed algorithm, so the speed-up factor is about 4 to 6 times at temporal level 2 which is higher than that of the temporal levels 0 and 1. Because the time for motion estimation used in temporal level 0 still occupies the largest portion in the whole motion estimation procedure, the speed-up factor of the motion estimation using the proposed algorithm is about 3 to 5 times as compared to the original algorithm. The initial estimated position is the approximation of the motion vector. Therefore, the resultant motion vector may be different from the motion vector found by the FSA leading to a slight degradation of the reconstructed PSNR quality. Figure 4.10, Figure 4.11 and Figure 4.12 depict the PSNR performance of the proposed algorithm for the “Coastguard”, “Foreman” and “Stefan” sequences respectively. The reconstructed PSNR values using the proposed algorithm are comparable to the original algorithm which does not use the median and motion vector predictions. The proposed algorithm can achieve a similar PSNR performance as compared with the FSA.



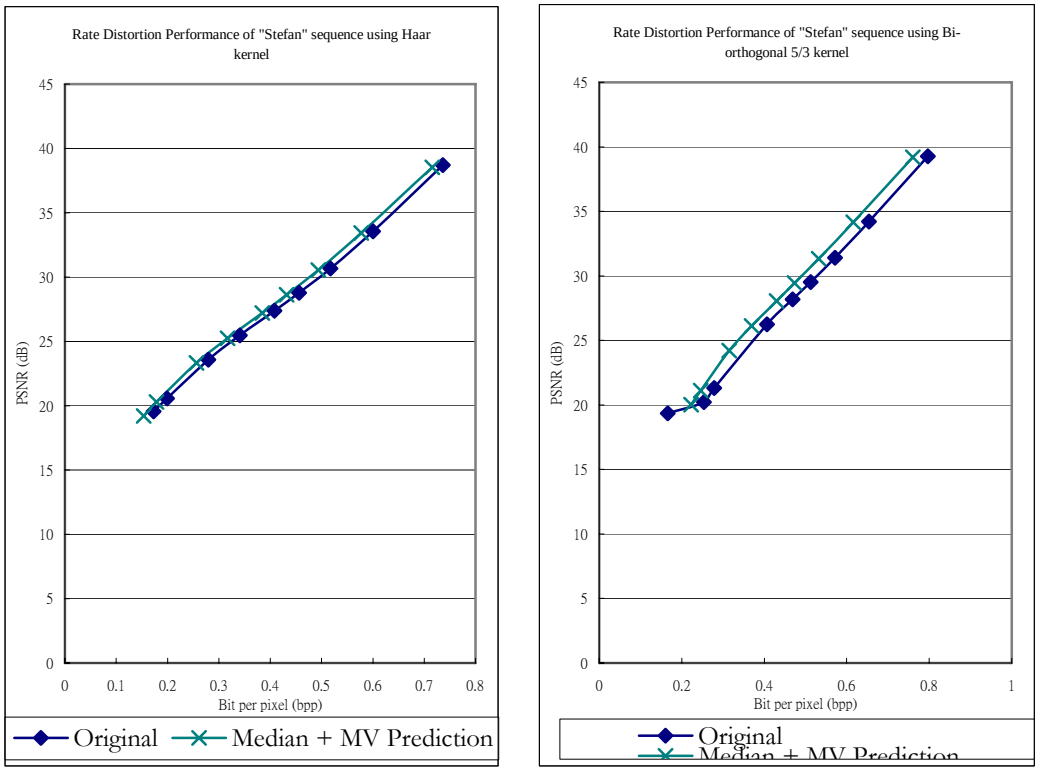
(a) **(b)**

Figure 4.10 Rate distortion performance of "Foreman" sequence for median prediction and cross-level motion vector prediction using (a) Haar kernel and (b) Bi-orthogonal 5/3 kernel during temporal decomposition



(a) **(b)**

Figure 4.11 Rate distortion performance of "Coastguard" sequence for median prediction and cross-level motion vector prediction using (a) Haar kernel and (b) Bi-orthogonal 5/3 kernel during temporal decomposition



(a) **(b)**

Figure 4.12 Rate distortion performance of "Stefan" sequence for median prediction and cross-level motion vector prediction using (a) Haar kernel and (b) Bi-orthogonal 5/3 kernel during temporal decomposition

Table 4.5 Total execution time used in motion estimation (ms) for median prediction and cross-level motion vector prediction using Haar kernel during temporal decomposition

Execution Time used for motion estimation (ms)	**Video sequence**							
	Trevor	**Suzie**	**Salesman**	**Grandmother**	**Foreman**	**Akiyo**	**Coastguard**	**Stefan**
level 0 (no med. + no mv pred. + FSA)	1516	1514	3813	3747	18235	17352	18017	14833
level 0 (no med. + no mv pred. + CPME-PDS)	475	442	926	1090	5145	3241	5012	4704
speed-up factor	**3.19**	**3.43**	**4.12**	**3.44**	**3.54**	**5.35**	**3.59**	**3.15**
level 1 (no med. + no mv pred. + FSA)	814	829	1889	1873	9170	8736	8939	7387
level 1 (no med. + no mv pred. + CPME-PDS)	250	239	468	541	2938	1622	2808	2886
speed-up factor	**3.26**	**3.47**	**4.04**	**3.46**	**3.12**	**5.39**	**3.18**	**2.56**
level 2 (no med. + no mv pred. + FSA)	421	372	921	1031	4545	4374	4470	3755
level 2 (no med. + no mv pred. + CPME-PDS)	106	92	202	205	1074	644	1078	1072
speed-up factor	**3.97**	**4.04**	**4.56**	**5.03**	**4.23**	**6.79**	**4.15**	**3.57**
Total (no med. + no mv pred. + FSA)	2751	2715	6623	6651	31950	30462	31426	41385
Total (no med. + no mv pred. + CPME-PDS)	831	773	1596	1836	9157	5507	8898	13549
speed-up factor	**3.31**	**3.51**	**4.15**	**3.62**	**3.49**	**5.53**	**3.53**	**3.05**
Execution Time used for temporal decomposition (ms)								
no med. + no mv pred. + FSA	2923	2982	7171	7188	34469	32717	33420	44297
med. + mv pred. + CPME-PDS	1093	1031	2202	2452	11811	8124	11515	16638
speed-up factor	**2.67**	**2.89**	**3.26**	**2.93**	**2.92**	**4.03**	**2.90**	**2.66**

Table 4.6 Total execution time used in motion estimation (ms) for median prediction and cross-level motion vector prediction using Bi-orthogonal 5/3 kernel during temporal decomposition

Execution Time used for motion estimation (ms)	**Video sequence**							
	Trevor	**Suzie**	**Salesman**	**Grandmother**	**Foreman**	**Akiyo**	**Coastguard**	**Stefan**
level 0 (no med. + no mv pred. + FSA)	2736	2766	6476	6798	31874	31034	32052	26345
level 0 (no med. + no mv pred. + CPME-PDS)	793	765	1640	1905	9025	5627	8721	8176
speed-up factor	**3.45**	**3.62**	**3.95**	**3.57**	**3.53**	**5.52**	**3.68**	**3.22**
level 1 (no med. + no mv pred. + FSA)	1203	1201	2891	2845	13750	13314	13536	11262
level 1 (no med. + no mv pred. + CPME-PDS)	374	363	701	843	4493	2458	4361	4314
speed-up factor	**3.22**	**3.31**	**4.12**	**3.37**	**3.06**	**5.42**	**3.10**	**2.61**
level 2 (no med. + no mv pred. + FSA)	406	345	788	890	4577	4461	4538	3778
level 2 (no med. + no mv pred. + CPME-PDS)	99	102	192	211	1105	643	1100	1059
speed-up factor	**4.10**	**3.38**	**4.10**	**4.22**	**4.14**	**6.94**	**4.13**	**3.57**
Total (no med. + no mv pred. + FSA)	4345	4312	10155	10533	50201	48809	50126	41385
Total (no med. + no mv pred. + CPME-PDS)	1266	1230	2533	2959	14623	8728	14182	13549
speed-up factor	**3.43**	**3.51**	**4.01**	**3.56**	**3.43**	**5.59**	**3.53**	**3.05**
Execution Time used for temporal decomposition (ms)								
no med. + no mv pred. + FSA	4705	4687	11029	11221	53690	52242	53574	44297
med. + mv pred. + CPME-PDS	1620	1596	3409	3824	18339	12389	17879	16638
speed-up factor	**2.90**	**2.94**	**3.24**	**2.93**	**2.93**	**4.22**	**3.00**	**2.66**

4.4 Conclusion

In this chapter, we propose a new motion estimation algorithm to reduce the computational complexity in the MCTF scheme. Since there exists large temporal correlation between successive video frames, so the average of two motion vectors in the previous temporal level can be used as an initial estimated position of the motion vector in the current temporal level. Then, the refinement process is performed in the reduced search window. Finally, the resultant motion vector is the vector sum of the initially estimated motion vector and a small refinement motion vector. Due to simplicity, we used the Haar kernel as an example to illustrate the concept of the proposed algorithm. The proposed idea can be applied to the kernel with longer filter length, such as Bi-orthogonal 5/3 kernel, in order to improve the compression efficiency. Besides, high spatial correlation exists in a video frame. Therefore, the median value of the motion vectors in the neighbouring blocks can be used as an initial estimated motion vector of the current block. Due to that initially estimated motion vector and the reduced search window, the computational complexity for motion estimation can be reduced significantly. The experimental results show that the time for motion estimation using the proposed algorithm is reduced by 3 to 5 times as compared with the FSA and the PSNR performance of the proposed algorithm is comparable to that of the FSA.

Chapter 5

Embedded Zerotree Wavelet (EZW)

5.1 Introduction

The Embedded Zerotree Wavelet (EZW) coding [57] is popular to be used to encode the wavelet coefficients due to its embedded nature and progressive transmission. It is an embedded coder because the encoded bitstream can achieve the target bit-rate by terminating the encoding procedure. Also, it can locate the coefficients with large magnitude so that they can be transmitted before the coefficients with small magnitude in order to achieve progressive transmission. Some modified EZW algorithms [58], [59], [60] are available to improve the coding gain by eliminating some less important wavelet coefficients in the high frequency subbands, such that the Human Visual System (HVS) cannot be aware of the degradation of the reconstructed image. This is because the HVS is more sensitive to the degradation of the low frequency components which correspond to the detail information, than that of the high frequency components which correspond to the edge information, in the natural image.

In this chapter, an algorithm is proposed to further improve the compression performance by discarding the wavelet coefficients in the high frequency subbands. The organization of this chapter is shown as follows. Some analysis of the EZW algorithm is discussed in section 5.2. The proposed algorithm is presented in section 5.3. Some experimental results are stated in section 5.4 to evaluate the performance of the proposed algorithm. Finally, a conclusion is drawn in the section 5.5.

5.2 Analysis of Embedded Zerotree Wavelet (EZW) algorithm

Some analysis of the conventional [57] and modified EZW algorithms using the minimum subband approach [60] are discussed in the sections 5.2.1 and 5.2.2 respectively.

5.2.1 Analysis of the conventional EZW algorithm

The EZW coding makes use of the bit-plane coding such that the most significant bit is firstly transmitted to the decoder in order to achieve progressive transmission. Each bit-plane refers to a pass. It can attain lossless coding by sending all passes to the decoder. If the information in last few passes is discarded, the reconstructed quality and bit per pixel (bpp) are both reduced. Although the distortion is introduced in the reconstructed image due to missing data in the last few passes, the human visual system cannot be aware by such distortion and the visual quality is not affected significantly. However, the compression efficiency is improved since the last few passes are not encoded.

Figure 5.1 and Figure 5.2 depict the reconstructed PSNR and bit per pixel of the conventional EZW algorithm for one, two and three decomposition levels stopping decoding at different passes using the D4 kernel and “Lena” image respectively. For using one decomposition level, the quality is the best but it requires more bits to encode the coefficients since the zerotree structure cannot be utilized efficiently. In the zerotree structure, one symbol, zerotree node, can represent a large portion of wavelet coefficients that are insignificant in the current pass. However, this structure cannot be used for using one decomposition level only. Figure 5.3 shows the rate distortion

performance of the EZW coding with different decomposition levels. The performance of three decomposition levels can outperform that of the two or one decomposition level(s) according to the efficient energy concentration to the low frequency subband. Furthermore, the zerotree only works efficiently for more than two decomposition levels since most energy can be concentrated to only a few number of wavelet coefficients for three or more decomposition levels. As a result, only a small number of symbols can represent many wavelet coefficients leading to increase the coding efficiency. Besides, the reconstructed “Lena” image using one, two and three decomposition levels for D4 kernel are illustrated in Figure 5.4, Figure 5.5 and Figure 5.6 respectively. When the last few passes are not decoded, the visual quality is not affected significantly. Since the data in the last few passes are insignificant, so the human visual system (HVS) cannot be aware of the degradation of omitting the information in these few passes. However, since some information in the last few passes are not encoded, it only attains the lossy coding. On the contrary, if all passes are encoded, it can achieve the lossless coding.

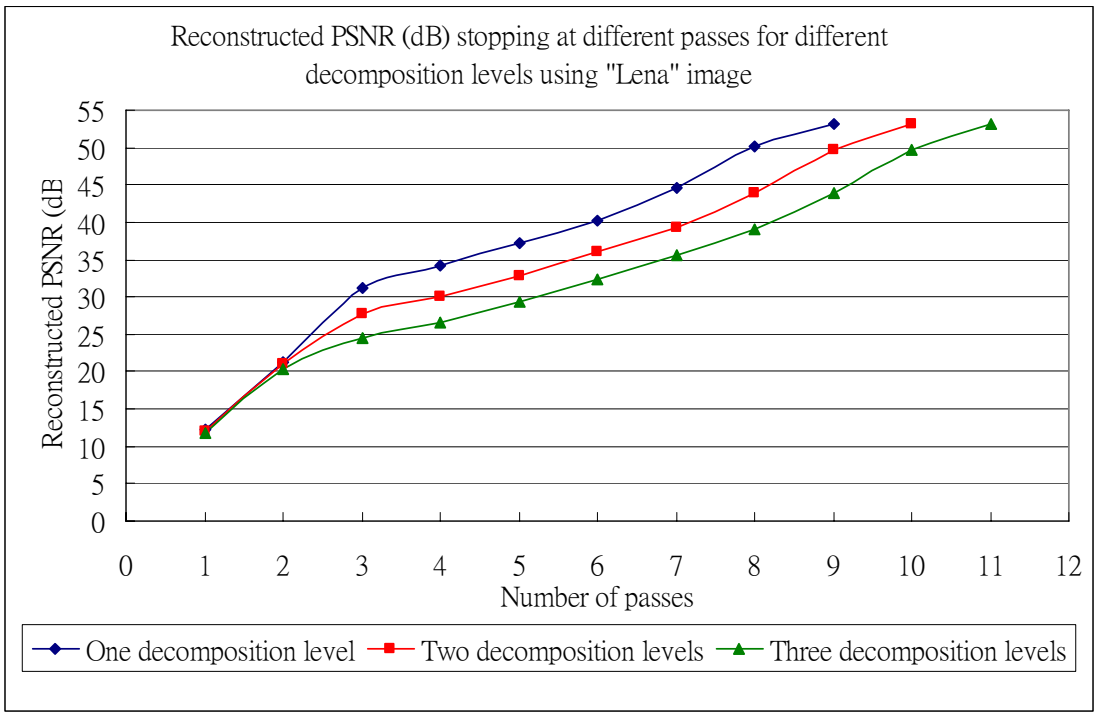


Figure 5.1 Reconstructed PSNR (dB) stopping at different passes for different decomposition levels using D4 kernel and "Lena" image

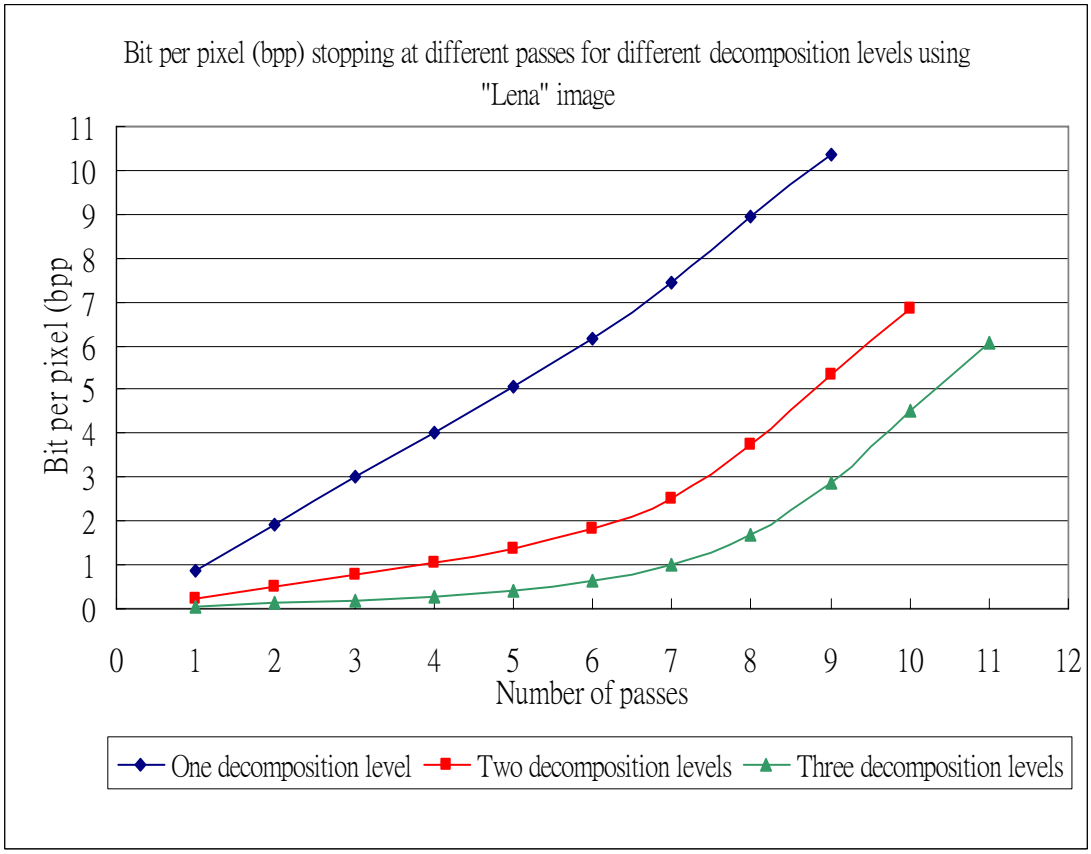


Figure 5.2 Bit per pixel (bpp) stopping at different passes for different decomposition levels using D4 kernel and "Lena" image

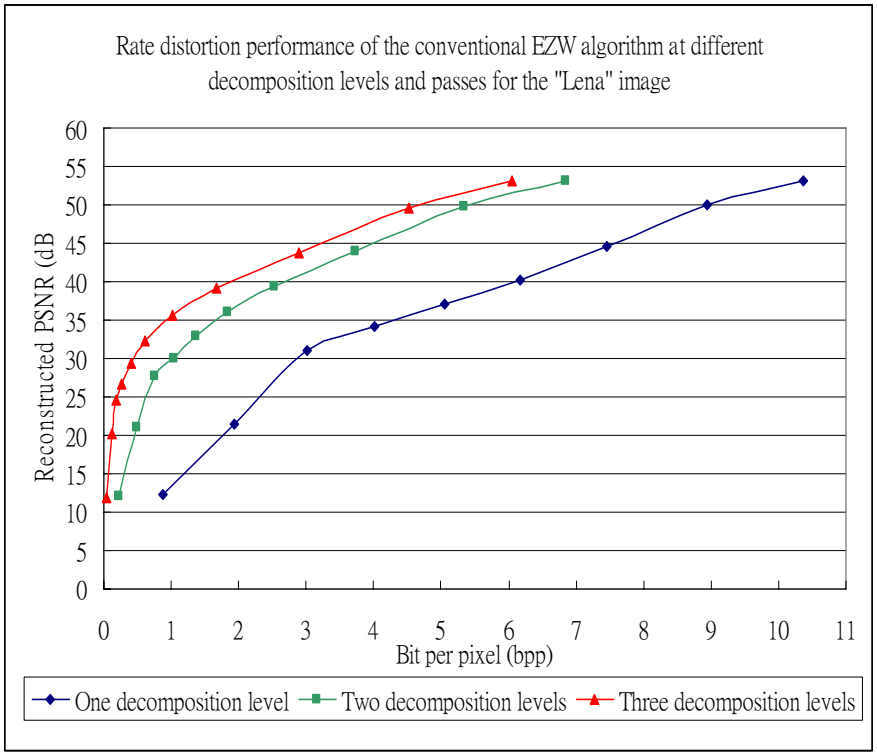


Figure 5.3 Rate distortion performance of the EZW coding with different decomposition levels using the D4 kernel for the Lena image

(a) (b) (c) (d) (e) (f)

Figure 5.4 Reconstructed images of the EZW algorithm with one decomposition level for (a) original image, decoding at (b) seven passes, (c) eight passes, (d) nine passes, (e) ten passes and (f) eleven passes (all passes) respectively

(a) (b)

(c) (d)

(e) (f)

Figure 5.5 Reconstructed images of the EZW algorithm with two decomposition levels for (a) original image, decoding at (b) six passes, (c) seven passes, (d) eight passes, (e) nine passes and (f) ten passes (all passes) respectively

(a) (b)

(c) (d)

(e) (f)

Figure 5.6 Reconstructed images of the EZW algorithm with three decomposition levels for (a) original image, decoding at (b) six passes, (c) seven passes, (d) eight passes, (e) nine passes and (f) ten passes (all passes) respectively

5.2.2 Analysis of the modified EZW algorithm using subband threshold approach

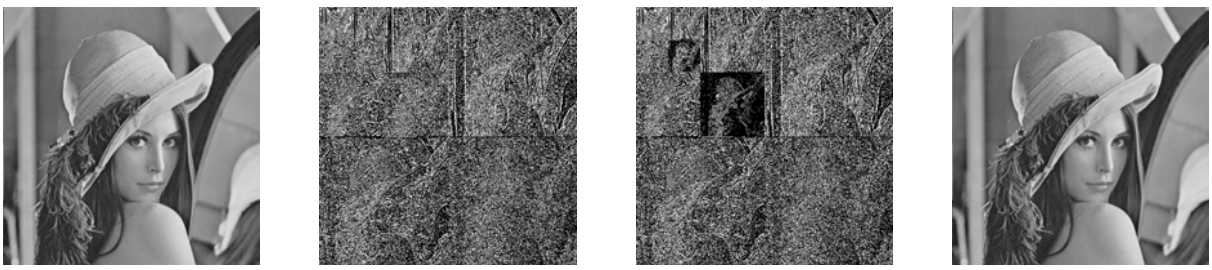
The information in the last few passes of the standard EZW algorithm is discarded in order to improve the coding gain and the HVS cannot be aware of the degradation by eliminating these less important data. However, this coding method is lossy such that the original information cannot be reconstructed. A modified EZW algorithm using the minimum weight subband approach [60] can further improve the compression efficiency by removing the wavelet coefficients in the high frequency subband if these coefficients are smaller than a pre-defined threshold. For this approach, the weight of a subband refers to the summation of all coefficients in that subband in magnitude. Then, a subband with the minimum weight in each decomposition level is selected. If the coefficients inside this minimum weight subband are smaller than a pre-determined threshold, these coefficients will be set to zero, i.e. eliminating these coefficients. However, there exists a problem that if there exists many coefficients with small magnitude and the large magnitude coefficients only occupy a small portion as compared with the small magnitude coefficients in a high frequency subband, there will be a probability that the subband with coefficients of small magnitudes will be selected as the minimum weight subband although a subband with large magnitude coefficient is available. Although the large magnitude coefficients can still be retained after performing thresholding in that subband, the compression efficiency is reduced as compared to perform thresholding in another subband which does not contain significantly high frequency coefficients. Therefore, we may add one more criterion to select the subband to perform thresholding. The extra criterion is the difference between the largest and smallest magnitude coefficients in each subband. This criterion can reflect the situation as described above. Table 5.1 and Table 5.3 show the minimum

weight and minimum difference subbands at each decomposition level using D4 and Daubechies 9/7 kernels respectively. For the “Lena” image, the minimum weight subband is the HH subband while the minimum difference subband is the LH subband in the level three as depicted in Table 5.1. For the “Fruit” image, the minimum weight subband is the LH subband while the minimum difference subband is the HH subband in levels one and three as illustrated in Table 5.3. These two Tables introduce a problem that these two criterion may indicate different subbands to perform thresholding and this problem will be solved in the section 5.4. Table 5.2 and Table 5.4 show the number of wavelet coefficients with zero magnitude for the conventional EZW algorithm and the modified EZW algorithm with minimum weight subband approach using D4 and Daubechies 9/7 kernels respectively. As expected, there exist coefficients with zero magnitude after performing the thresholding process. Also, when the threshold becomes larger, more coefficients are set to zero leading to an increase in coding gain but a reduction in reconstructed quality. As a result, there is a trade-off between the compression ratio and the reconstructed quality. However, the visual quality of the reconstructed image is not notably affected as shown in Figure 5.7 and Figure 5.9. The HVS is not sensitive to the threshold values between two and five.

(a) (b)

(c) (d)

Figure 5.7 Reconstructed images of the (a) original image, (b) conventional EZW algorithm, (c) modified EZW algorithm with pre-processing of the value of threshold of two and (d) modified EZW algorithm with pre-processing of the value of threshold of five respectively with three decomposition levels using D4 kernel



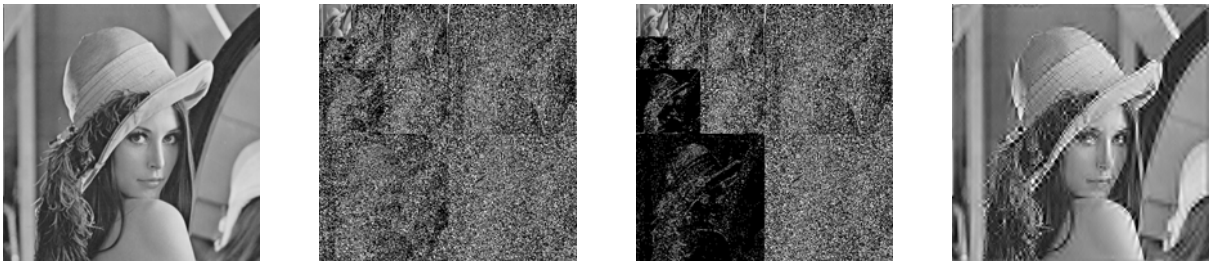
(a) (b) (c) (d)

Figure 5.8 (a) Original image, (b) wavelet-transformed image, (c) pre-processed image with a threshold of five and (d) reconstructed image with three decomposition levels using a threshold of five

(a) (b)

(c) (d)

Figure 5.9 Reconstructed images of the (a) original image, (b) conventional EZW algorithm, (c) modified EZW algorithm with pre-processing of the value of threshold of two and (d) modified EZW algorithm with pre-processing of the value of threshold of five respectively with three decomposition levels using Daubechies 9/7 kernel



(a) (b) (c) (d)

Figure 5.10 (a) Original image, (b) wavelet-transformed image, (c) pre-processed image with a threshold of five and (d) reconstructed image with three decomposition levels using a threshold of five

Table 5.1 Reconstructed quality (dB) and the minimum weight subband and minimum difference subband in each level for the conventional EZW algorithm and the modified EZW algorithm with pre-processing stage using the three decomposition levels and D4 kernel

Image	Conventional EZW algorithm	Min. Weight / Min. Different			Modified EZW algorithm with pre-processing	
		Level 1	Level 2	Level 3	Threshold = 2	Threshold = 5
Lena	53.1866 dB	HH / HH	HH / HH	HH / LH	52.841 dB	50.434 dB
Fruit	53.1473 dB	HH / HH	HH / HH	HH /HH	51.8463 dB	45.5136 dB

Table 5.2 The number of wavelet coefficients with zero magnitude for the conventional EZW algorithm and the modified EZW algorithm with pre-processing stage using the three decomposition levels and D4 kernel

Image	Number of zero coefficients for conventional EZW algorithm	Number of zero coefficients for modified EZW algorithm with pre-processing	
		Threshold = 2	Threshold = 5
Lena	0	7451	13412
Fruit	0	30627	62010

Table 5.3 Reconstructed quality (dB) and the minimum weight subband and minimum difference subband in each level for the conventional EZW algorithm and the modified EZW algorithm with pre-processing stage using the three decomposition levels and Daubechies 9/7 kernel

Image	Conventional EZW algorithm	Min. Weight / Min. Different			Modified EZW algorithm with pre-processing	
		Level 1	Level 2	Level 3	Threshold = 2	Threshold = 5
Lena	24.5371 dB	LH / LH	LH / LH	LH / LH	24.5404 dB	24.577 dB
Fruit	24.948 dB	LH / HH	LH / LH	LH / HH	24.9494 dB	24.9617 dB

Table 5.4 The number of wavelet coefficients with zero magnitude for the conventional EZW algorithm and the modified EZW algorithm with pre-processing stage using the three decomposition levels and Daubechies 9/7 kernel

Image	Number of zero coefficients for conventional EZW algorithm	Number of zero coefficients for modified EZW algorithm with pre-processing	
		Threshold = 2	Threshold = 5
Lena	0	46387	72935
Fruit	0	8767	13428

5.3 Proposed algorithm of the modified EZW algorithm

The key idea of the proposed algorithm is to eliminate the insignificant coefficients in the high frequency subbands, i.e. HL, LH and HH subbands. According to the observation in the previous section, one more criterion, which is the minimum difference, is used to choose the subband to discard the unimportant coefficients, where the minimum difference is the absolute difference between the coefficients with the

greatest and smallest magnitudes. If the minimum weight subband [60] is the same as the minimum difference subband, we can perform further quantization in that subband because that subband must contain insignificant information such that the human visual system cannot be aware by discarding such insignificant data. Otherwise, we perform thresholding in the minimum weight subband such that the significant information can still be retained. The procedure of the proposed algorithm is summarized as follows.

1. Find the minimum weight subband in each decomposition level, where the weight of each subband is the sum of all coefficients in magnitude.

2. Find the minimum difference subband in each decomposition level, where the difference of each subband is the difference between the largest and smallest magnitude of the coefficients.

3. At each decomposition level, if the minimum weight subband is the same as the minimum difference subband, all coefficients in that subband are quantized by a specified quantizer. Otherwise, for each coefficient in the minimum weight subband, if the magnitude of coefficient is less than a pre-determined threshold value, it will be set to zero.

The processed wavelet coefficients are then put forward to entropy encoding process. The compressed bitstream is conveyed to the decoder for reconstruction. As the insignificant wavelet coefficients in the high frequency subband are discarded, the compression efficiency can be further enhanced. Besides, the visual quality of the reconstructed image is not considerably affected. The proposed algorithm is a lossy coding scheme since some unimportant information in the high frequency subbands is eliminated and they cannot be recovered during reconstruction.

As mentioned in section 2.6, the Set-Partition Embedded Block Coder (SPECK) algorithm [107] is the latest coding algorithm to encode the wavelet coefficients. The

proposed approach can also be applied in the SPECK algorithm. The major difference between the EZW algorithm and the SPECK algorithm is that the SPECK algorithm scans the wavelet coefficients in depth-first scanning path while the EZW algorithm uses the transversal scanning path. For the SPECK algorithm, if a set of coefficients are tested to be significant, then it is split into four equal subsets. Each subset is carried significant test until the significant coefficients can be found out. Therefore, the scanning path of SPECK algorithm refers to the depth-first searching path. For the EZW algorithm, if a coefficient including all of its descendents are tested to be significant, then a symbol, IZ, isolated zero is put to the dominant list. Then, the coefficients in the same subband and the same decomposition levels are scanned. After that, the descendents of significant coefficient are scanned. As a result, the EZW algorithm makes use of the transversal searching path. Due to this major difference, the procedure of proposed algorithm is modified in the follows so that it can take the advantage of the SPECK algorithm.

1. Find the minimum weight subband in each decomposition level, where the weight of each subband is the sum of all coefficients in magnitude.

2. Find the minimum difference subband in each decomposition level, where the difference of each subband is the absolute difference between the largest and smallest magnitude of the coefficients.

3. At each decomposition level, if the minimum weight subband is the same as the minimum difference subband, the magnitude of coefficient smaller than a pre-defined threshold value in that subband are quantized by a specified quantizer. Otherwise, for each coefficient in the minimum weight subband, if the magnitude of coefficient is less than a pre-determined threshold value, it will be set to zero.

The processed wavelet coefficients are then put forward to SPECK and arithmetic encoding procedures. The compressed bitstream is sent to the decoder for reconstruction. As the insignificant wavelet coefficients in the high frequency subband are removed, the compression efficiency can be further enhanced. Besides, the visual quality of the reconstructed image is considerably affected. The proposed algorithm is a lossy coding scheme since some less important information in the high frequency subbands is removed and they cannot be recovered during reconstruction.

5.4 Experimental Results

The “Lena” image was used to evaluate the performance of the proposed algorithm. The D4 kernel with three decomposition levels was performed during the encoding and decoding processes. After carrying out the EZW coding, the Huffman coding is implemented to convert the symbols of the EZW coding into compressed bitstream which were sent to the decoder for reconstruction.

The rate distortion performance of the conventional EZW algorithm, modified EZW method using the minimum subband approach and proposed algorithm using the “Lena” and “Fruit” images are depicted in Figure 5.12 and Figure 5.13 respectively. The proposed algorithm can both outperform the conventional EZW algorithm and the minimum subband approach due to discarding the unimportant wavelet coefficients in the high frequency subband at each decomposition level. Therefore, the number of bits used to encode the coeffcieints can be reduced leading to decreasing the number of bits per pixel (bpp) at the same reconstructed quality. There exists a probability that the minimum subband approach may select the significant subband to perform thresholding. As a result, one more criterion, which is the minimum difference subband, is used to

choose the most insignificant subband. If both the minimum subband and minimum difference subbands indicate the same high frequency subband, this subband must be the most insignificant in certain decomposition level. So, we can remove these insignificant coefficients in that unimportant subband to improve the coding efficiency, and there is insignificant effect on the visual quality. Figures 5.11 (c) and (d) show the visual quality of the reconstructed image using the minimum subband approach and the proposed algorithm respectively. Since only insignificant wavelet coefficients are eliminated in the high frequency subband and our human eyes are not sensitive to the distortion of these insignificant information, the visual quality of the reconstructed image are essentially not affected.

(a) (b)

(c) (d)

Figure 5.11 Reconstructed images of the (a) original image, (b) conventional EZW algorithm, (c) modified EZW algorithm with pre-processing of the value of threshold of two and (d) proposed EZW algorithm with pre-processing of the value of threshold of two and the quantization factor of two respectively with three decomposition levels using D4 kernel

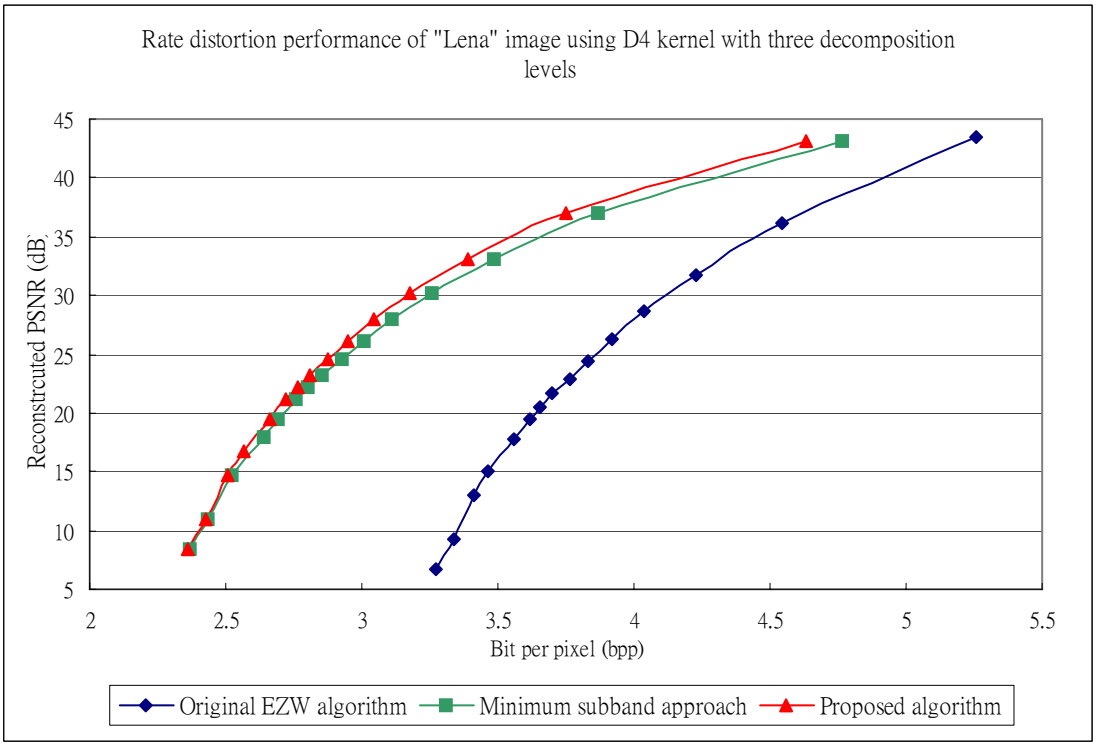


Figure 5.12 Rate distortion performance of the conventional EZW algorithm, modified EZW algorithm using the minimum weight subband approach and the proposed EZW algorithm using the D4 kernel with three decomposition levels for the "Lena" image

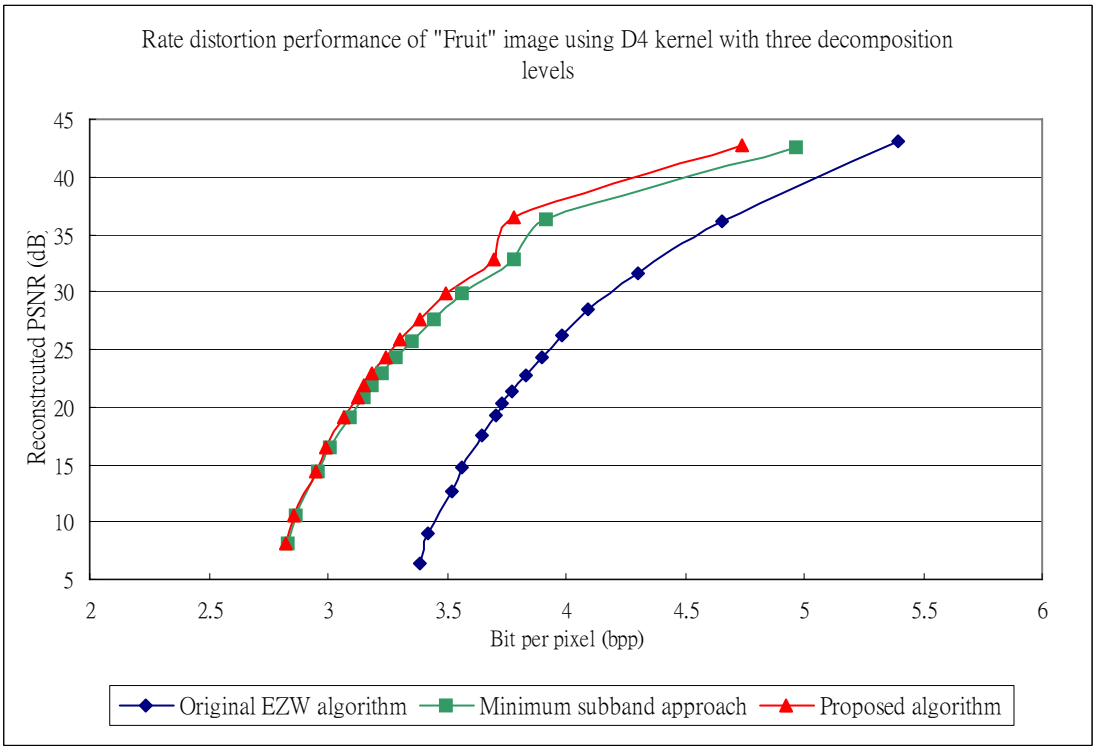


Figure 5.13 Rate distortion performance of the conventional EZW algorithm, modified EZW algorithm using the minimum weight subband approach and the proposed EZW algorithm using the D4 kernel with three decomposition levels for the "Fruit" image

The rate distortion performance of the conventional SPECK algorithm [107], modified SPECK method using the minimum subband approach and proposed approach employed in the SPECK algorithm using D4 kernel with three decomposition levels for the “Lena” and “Fruit” images are depicted in Figure 5.15 and Figure 5.16 respectively. According to the experimental results, the proposed approach can outperform the minimum subband approach. It can select and remove some unimportant coefficients in the high frequency subbands at each decomposition level. After eliminating the insignificant coefficients, the number of bits used to encode the coefficients are reduced and the coding efficiency can be improved as compared to the minimum subband approach.

(a) (b)

(c) (d)

Figure 5.14 Reconstructed images of the (a) original image, (b) conventional SPECK algorithm, (c) modified EZW algorithm with pre-processing of the value of threshold of five and (d) proposed SPECK algorithm with pre-processing of the value of threshold of five and the quantization factor of two respectively with three decomposition levels using D4 kernel

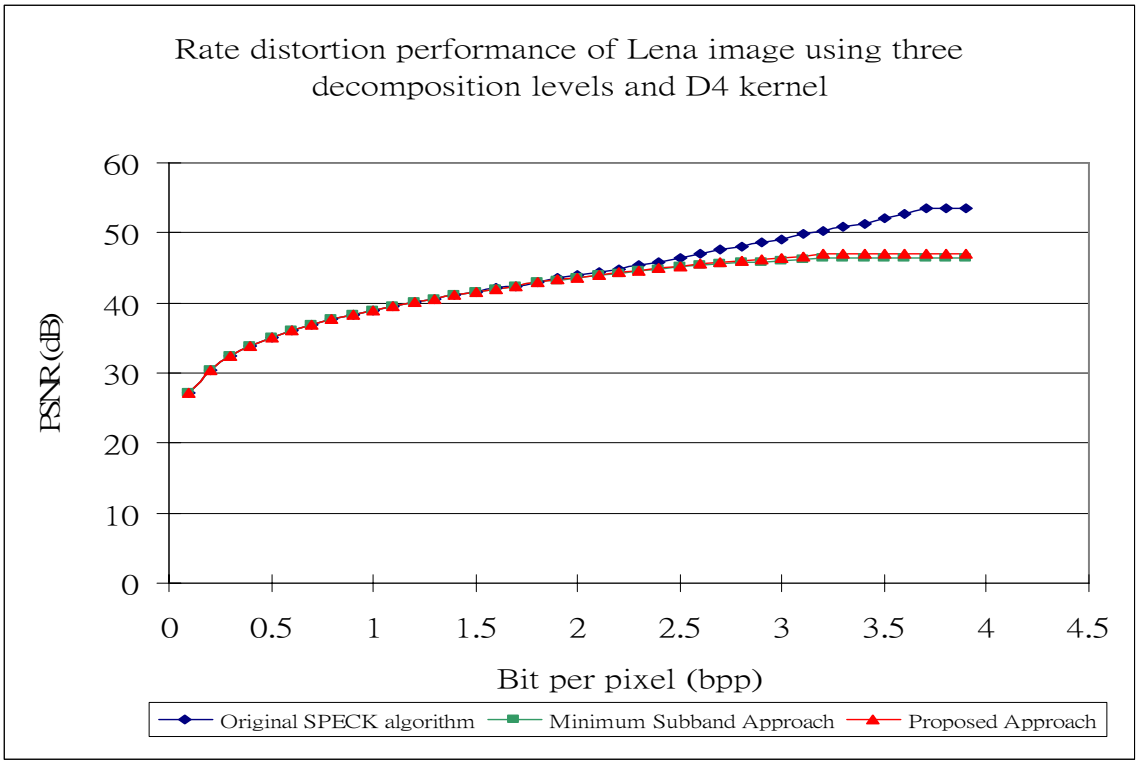


Figure 5.15 Rate distortion performance of the conventional SPECK algorithm, modified SPECK algorithm using the minimum weight subband approach and the proposed SPECK algorithm using the D4 kernel with three decomposition levels for the "Lena" image

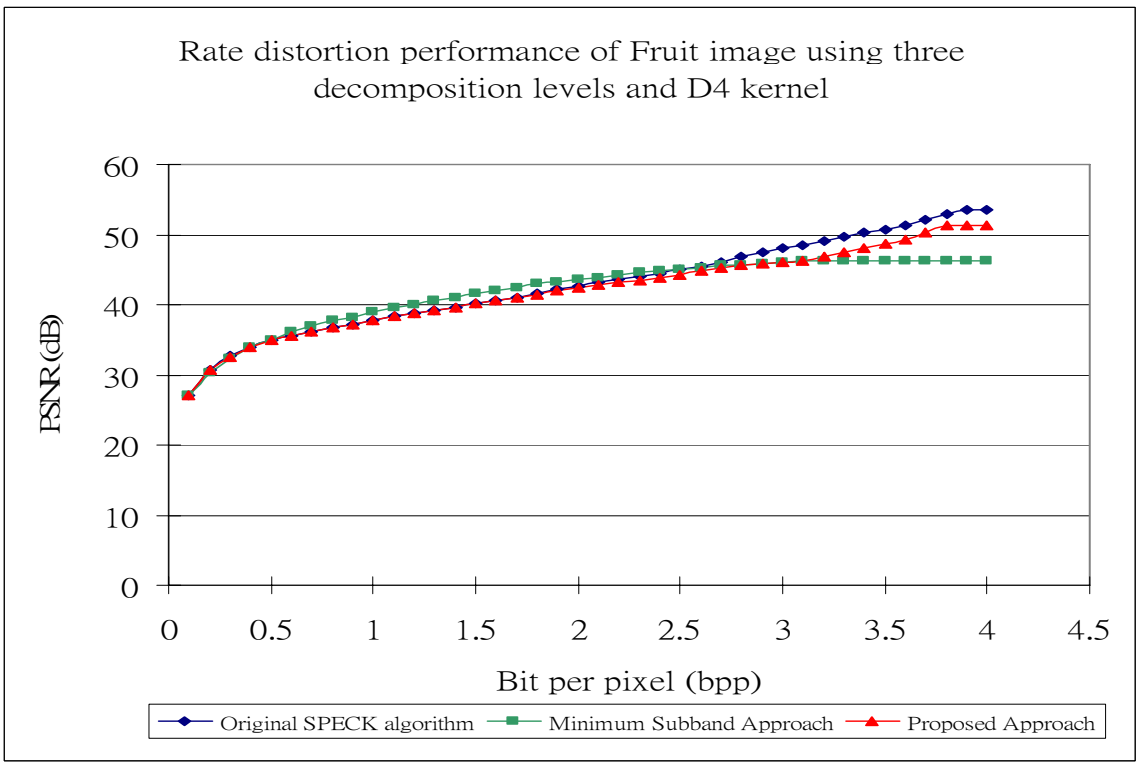


Figure 5.16 Rate distortion performance of the conventional SPECK algorithm, modified SPECK algorithm using the minimum weight subband approach and the proposed SPECK algorithm using the D4 kernel with three decomposition levels for the "Fruit" image

5.5 Conclusion

In this chapter, we propose a modified EZW algorithm to improve the compression efficiency in image compression. Since there exist a large amount of insignificant wavelet coefficients in the high frequency subband and this information consumes bits to encode, these less important coefficients should be systematically discarded in order to improve the coding efficiency. Furthermore, the HVS is not sensitive to the degradation introduced from this unimportant information. Therefore, the visual quality is not considerably affected by eliminating such insignificant wavelet coefficients. Two criteria, which are the minimum weight and minimum difference, are used to select the insignificant high frequency subband in each decomposition level. The weight of each subband is the summation of all wavelet coefficients in magnitude while the difference of each subband is the absolute difference between the greatest and smallest wavelet coefficients. If the minimum weight subband is the same as the minimum difference subband, all wavelet coefficients in this high frequency subband perform the quantization using a pre-defined quantization step-size. Otherwise, the less important wavelet coefficients, which are smaller than a specified threshold, in the minimum weight subband are discarded. As the number of bits used to encode the wavelet coefficients is reduced, the coding gain can be increased.

From the experimental results, the average number of bits used to encode each pixel of the proposed algorithm is reduced by 0.1 bit per pixel (bpp) and 1 bpp as compared with the minimum subband approach [60] and conventional EZW algorithm [57] respectively with similar PSNR values. Besides, the visual quality of the reconstructed image using the proposed algorithm is comparable to that of the

conventional EZW algorithm. In addition, the proposed algorithm can also be applied to the latest coding scheme, the SPECK algorithm [107], and can outperform the minimum subband approach [60] by 0.1 bit per pixel (bpp) with the same PSNR value. The visual quality can still be maintained as compared to the original SPECK algorithm and minimum subband approach.

Chapter 6

Conclusion

6.1 Conclusion on the current works

In this work, the typical wavelet video coder is studied. Some fast algorithms of motion estimation in the wavelet domain are proposed and some conclusions are drawn in this chapter.

The fundamental concept of the hybrid video coding model is reviewed in Chapter 2 and this model is employed in the traditional video coding standards such as MPEG-1, MPEG-2, MPEG-4 and H.264. The classic wavelet video coding system also makes use of the hybrid video coding model to reduce both spatial and temporal redundancies in the video sequences. There are two major differences between the wavelet video coding system and conventional hybrid video coding model. The first one is the transform kernel. The wavelet transform is used in order to achieve superior compression performance and eliminate the blocking artefacts as compared with the conventional transform kernel, i.e. the Discrete Cosine Transform (DCT). Also, the wavelet transform is scalable in nature so that it can be used in the multi-resolution applications such as Digital TV (DTV) and High-Definition TV (HDTV). The other difference is that the motion estimation is performed in the wavelet domain. As there exist high correlations between the corresponding subbands across different decomposition levels in the wavelet pyramid, so the speed of motion estimation can be enhanced by exploiting such property. Several typical wavelet-domain motion

estimation algorithms are also studied in the chapter 2. Since the two dimensional Discrete Wavelet Transform (2D-DWT) is used in the standard wavelet video coder, so the spatial scalability can only be achieved but not the temporal scalability. Hence, the three dimensional Discrete Wavelet Transform (3D-DWT) is performed in the novel wavelet video coder in order to achieve both temporal and spatial scalabilities by decomposing the video frames in the temporal direction along the motion trajectories. Thus, the visual quality of the low frequency frame is improved by reducing the ghosting artifacts. Besides, the compression efficiency can be improved as compared with the 2D-DWT video coder. And the architecture of 3D-DWT video coder is revised in the chapter 2. After carrying out the wavelet transform, the wavelet coefficients are encoded by the Embedded Zerotree Wavelet (EZW) algorithm by exploiting the redundancy existing among different subbands. The EZW algorithm is also investigated in the chapter 2.

A fast motion estimation algorithm in the wavelet domain in the 2D-DWT video encoder is proposed in the chapter 3. The pixel error with similar magnitude tends to group in clusters in the spatial domain. The Clustered Pixel Matching Error for Partial Distortion Search (CPME-PDS) is investigated to exploit such clustering property in the spatial domain in order to improve the speed of the motion estimation. According to our observation, this pixel cluster property is also available in the wavelet domain. Besides, this clustering property is appeared in the hierarchical nature of the wavelet pyramid. By applying the hierarchical property of the wavelet domain into the row-based CPME-PDS, the impossible candidate blocks can be rejected as early as possible. Only the LL subband and the subbands in the lowest resolution level have to carry out the counting sort to find their error distributions. The results can then be down-sampled and re-numbered in the subband at higher resolution level of the hierarchical pyramid.

Experimental results show that the proposed scheme, i.e. the backward CPME-PDS algorithm, can improve the speed of motion estimation in terms of total execution time and the average number of operations per block comparing to Full Search Algorithm (FSA) in Multi-Resolution Motion Estimation (MRME) scheme. By applying the Successive Elimination Algorithm (SEA) into the backward CPME-PDS, the average number of operations per block can be further reduced. However, its implementation time is longer than that of the backward CPME-PDS due to greater number of comparisons of the floating-point numbers (wavelet coefficients) and the structure of CPU is not favour to floating-point number comparison. But the performance of proposed algorithm in the slow motion video sequences is extremely well. Hence, it is suitable for video conferencing and video surveillance. Due to the scalability nature of the Discrete Wavelet Transform (DWT), the proposed scheme can be applied to multi-resolution applications such as Digital TV (DTV), High Definition TV (HDTV) and mobile phone applications.

In chapter 4, we propose a new motion estimation algorithm to reduce the computational complexity in the 3D-DWT video encoder. Since there exists large temporal correlation between successive video frames, so the average of two motion vectors in the previous temporal level can be used as an initial estimated position of the motion vector in the current temporal level. Then, the refinement procedure is carried out in the reduced search window. Finally, the resultant motion vector is the vector sum of the initially estimated motion vector and a small refinement motion vector. The proposed idea can be applied to the kernel with longer filter length, such as Bi-orthogonal 5/3 kernel, in order to improve the compression efficiency. Besides, high spatial correlation exists in a video frame. Therefore, the median value of the motion vectors in the neighbouring blocks can be used as an initial estimated motion vector of

the current block in the first temporal level, i.e. the spatial domain. Due to that initially estimated motion vector and the reduced search window, the computational complexity for motion estimation can be reduced significantly. The experimental results show that the execution time and average number of operations per block for motion estimation using the proposed algorithm are both reduced as compared with the FSA. But the PSNR performance of the proposed algorithm can be retained and it is comparable to PSNR performance of the FSA.

In chapter 5, we propose a modified EZW algorithm to improve the compression efficiency in image compression. Since there exist a large amount of insignificant wavelet coefficients in the high frequency subband and this information requires lots of bits to encode, so these less important coefficients can be discarded in order to improve the coding efficiency. Furthermore, the Human Visual System (HVS) is not sensitive to the degradation introduced from this unimportant information. Therefore, the visual quality is not considerably affected by discarding such insignificant wavelet coefficients. Two criteria, which are the minimum weight and minimum difference, are used to select the insignificant high frequency subband in each decomposition level. The weight of each subband is the summation of all wavelet coefficients in magnitude while the difference of each subband is the absolute difference between the greatest and smallest wavelet coefficients. If the minimum weight subband is the same as the minimum difference subband, all wavelet coefficients in this high frequency subband performs the quantization using a pre-defined quantization step-size. Otherwise, the less important wavelet coefficients, which are smaller than a specified threshold, in the minimum weight subband are discarded. As the number of bits used to encode the wavelet coefficients is reduced, so the coding gain can be increased. From the experimental results, the average number of bits used to encode each pixel of the proposed algorithm

is reduced as compared with the modified EZW algorithm with minimum subband approach and the conventional EZW algorithm for similar PSNR performance. Besides, the visual quality of the reconstructed image using the proposed algorithm can be preserved and it is comparable to reconstructed quality of the conventional EZW algorithm.

6.2 Future research directions

There are some future research directions for the motion estimation in both 2D-DWT and 3D-DWT video encoders.

For the motion estimation algorithm in the 2D wavelet domain, the proposed algorithm (backward CPME-PDS) makes use of the row-based computation to obtain the partial Sum of Absolute Difference (SAD) row by row. The HL subband can take advantage of this row-based computation strategy as the horizontal edge exists in the HL subband. On the other hand, the partial SAD calculation in the LH subband can be performed in column-by-column strategy since the vertical edge is located in this subband. By exploiting the edge property of the wavelet subband, the speed of the motion estimation can be further enhanced.

The latest video coding standard, H.264, makes use of the variable block size motion estimation with rate-distortion optimization scheme to further enhance the compression efficiency at the expense of computational complexity for motion estimation. This approach can be applied to the 3D-DWT video coder to improve the coding gain. However, the computational burden for motion estimation is increased significantly. Thus, some fast variable block size motion estimation algorithms for the wavelet video encoder can be investigated to optimize the tradeoff between the compression efficiency and the computational load.

Furthermore, the current Motion Compensated Temporal Filtering (MCTF) scheme in the 3D-DWT video coder requires decoding the motion vectors in all temporal levels even though only half temporal resolution of video sequence is decoded, say for example. Thus, the scalable motion vector coding approach should be investigated. If a quarter of the original temporal resolution is decoded, then the motion vectors of the quarter temporal resolution are decoded in order to avoid full decoding of motion vectors in all temporal resolution levels.

References

[1]. ISO/IEC 11172-2, “Information technology – coding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbits/s – part 2: video”, ISO/IEC 11172-2 (MPEG-1), March 1993.

[2]. ITU-T and ISO/IEC, “Information technology – generic coding of moving pictures and associated audio information: video”, ITU-T Recommendation H.262 – ISO/IEC 13818-2 (MPEG-2), November 1994.

[3]. ISO/IEC, “Information technology – coding of audio-visual objects: Part 2 visual”, ISO/IEC 14496-2 (MPEG-4), October 1998.

[4]. ITU-T, “Video coding for low bit rate communication”, ITU-T Recommendation H.263, Version 1, November 1995; Version 2, January 1998.

[5]. Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification (ITU-T Rec. H.264 | ISO/IEC 14496-10 AVC), July 2003.

[6]. Thomas Wiegand, Gary J. Sullivan, Gisle Bjøntegarrd and Ajay Luthra, “Overview of the H.264/AVC Video Coding Standard”, IEEE Transactions on Circuits and Systems for Video Technology, Vol. 13, No. 7, July 2003.

[7]. Yuichiro Nakaya and Hiroshi Harashima, “Motion Compensation based on Spatial Transformations”, IEEE Transactions on Circuits and Systems for Video Technology, Vol. 4, pp. 339- 356, June 1994.

[8]. M. Ghanbari, S. de Faria, I. N. Goh and K. T. Tan, “Motion Compensation for Very Low Bit-Rate Video”, Signal Processing: Image Communication, Vol. 7, pp. 567 – 580, 1995.

[9]. Bernd Girod, “The Efficiency of Motion-Compensating Prediction for Hybrid Coding of Video Sequences”, IEEE Journal on Selected Areas in Communications, Vol. SAC-5, No. 7, pp. 1140 – 1154, August 1987.

[10]. Bernd Girod, “Motion-Compensation Prediction with Fractional-Pel Accuracy”, IEEE Transactions on Communications, Vol. 41, pp. 604 – 612, April 1993.

[11]. S. Nogaki and M. Ohta, “An Overlapped Block Motion Compensation for High Quality Motion Picture Coding”, Proceedings of International Symposium on Circuits and Systems, Vol. 1, pp. 184 – 187, May 1992.

[12]. Chang-Da Bei and Robert M. Gray, “An Improvement of the Minimum Distortion Encoding Algorithm for Vector Quantization”, IEEE Transactions on Communications, Vol. Com-33, No. 10, October 1985.

[13]. Chun-Ho Cheung and Lai-Man Po, “Adjustable Partial Distortion Search Algorithm for Fast Block Motion Estimation”, IEEE Transactions on Circuits and Systems for Video Technology, Vol. 13, Issue 1, pp. 100 – 110, January 2003.

[14]. Yui-Lam Chan and Wan-Chi Siu, “An Adaptive Partial Distortion Search for Block Motion Estimation”, IEEE Proceedings of International Conference on Acoustics, Speech and Signal Processing, Vol. 3, pp. 153 – 156, April 2003.

[15]. Chun-Ho Chung and Lai-Man Po, “Generalized Partial Distortion Search Algorithm for Block-Matching Motion Estimation, IEEE Proceedings on International Conference on Image Processing, Vol. 3, pp. 510 – 513, October 2001.

[16]. Zi-Yin Li and Shan-An Zhu, “A Fast and Efficient Partial Distortion Search For Block Motion Estimation, IEEE Proceedings on International Conference on Control and Automation, Vol. 1, pp. 234 – 238, June 2005.

[17]. Chi-Wang Ting, Chi-Wai Lam and Lai-Man Po, "A Novel Early-Accepted Partial Distortion Search Algorithm for Block Motion Estimation, IEEE Proceedings on International Symposium on Intelligent Multimedia, Video and Speech Processing, pp. 414 – 417, October 2004.

[18]. Ko-Cheung Hui, Wan-Chi Siu and Yui-Lam Chan, "New Adaptive Partial Distortion Search Using Clustered Pixel Matching Error Characteristic", IEEE Transactions on Image Processing, Vol. 14, Issue 5, pp. 597 – 607, May 2005.

[19]. Xiaoquan Yi and Nam Ling, "Improved Partial Distortion Search Algorithm for Rapid Block Motion Estimation via Dual-Halfway-Stop, IEEE Proceedings on Acoustics, Speech and Signal Processing, Vol. 2 pp. 917 – 920, March 2005.

[20]. Chok-Kwan Cheung and Lai-Man Po, "Normalized Partial Distortion Search Algorithm for Block Motion Estimation", IEEE Transactions on Circuits and Systems for Video Technology, Vol. 10, Issue 3, pp. 417 – 422, April 2000.

[21]. Chun-Ho Cheung and Lai-Man Po, "A Fast Block Motion Estimation Using Progressive Partial Distortion Search", IEEE Proceedings on Intelligent Multimedia, Video and Signal Processing, pp. 506 – 509, May 2001.

[22]. Won-Gi Hong and Tae-Myong Oh, "Sorting-Based Partial Distortion Search Algorithm for Motion Estimation", Electronics Letters, Vol. 40, Issue 2, pp. 113 – 115, January 2004.

[23]. Won-Gi Hong and Tae-Myong Oh, "Enhanced Partial Distortion Search Algorithm for Block Motion Estimation", Electronics Letters, Vol. 39, Issue 15, pp. 1112 – 1113, July 2003.

[24]. Wenbin Jiang and Manli Zhou, "A Fast BMA Based on Combined Search Candidate Subsampling and APDS", IEEE International Conference on Multimedia and Expo, Vol. 2, pp. 1115 – 1118, June 2004.

[25]. Chok-Kwan Cheung and Lai-Man Po, "A Hierarchical Block Matching Algorithm using Partial Distortion Measure", IEEE Proceedings on Circuits and Systems, Vol. 2, pp. 1237 – 1240, June 1997.

[26]. Yui-Lam Chan and Wan-Chi Siu, "Edge Oriented Block Motion Estimation for Video Coding, IEE Proceedings on Vision, Image and Signal Processing, Vol. 114, Issue 3, pp. 136 – 144, June 1997.

[27]. Yui-Lam Chan and Wan-Chi Siu, "An Efficient Search Strategy for Block Motion Estimation using Image Feature", IEEE Transactions on Image Processing, Vol. 10, Issue 8, pp. 1223 – 1238, August 2001.

[28]. Yui-Lam Chan, Ko-Cheung Hui and Wan-Chi Siu, "Fast Search Algorithm for Edge-Oriented Block Matching Algorithm, IEEE Proceedings of International Intelligent Multimedia, Video and Speech Processing, pp. 225 – 228, May 2001.

[29]. M. B. Ahmad, Kong Yoon Kim, Kyong Sig Roh and Taw Sun Choi, "Motion Vector Estimation using Edge Oriented Block Matching Algorithm for Video Sequences, IEEE Proceedings on International Conference on Image Processing, Vol. 1, pp. 860 – 863, September 2000.

[30]. W. Li and E. Salari, "Successive Elimination Algorithm for Motion Estimation", IEEE Transactions on Image Processing, Vol. 4, No. 1, pp.105 – 107, January 1995.

[31]. Hung-Sheng Wang, R. M. Mersereau, "Fast Algorithm for the Estimation of Motion Vectors", IEEE Transactions of Image Processing, Vol. 8, No. 1, pp. 435 – 438, March 1999.

[32]. Soo-Mok Jung, Sung-Chul Shin, Hyunki Baik and Myong-Soon Park, "Nobel Successive Elimination Algorithm for the Estimation of Motion Vectors", IEEE Proceedings of International Symposium on Multimedia Software Engineering, pp. 332 – 335, December 2000.

[33]. Soo-Mok Jung, Sung-Chul Shin, Hyunki Baik and Myong-Soon Park, "New Fast Successive Eliminiation Algorithm", IEEE Proceedings of 43rd Midwest Symposium on Circuits and Systems, Vol. 2, pp. 616 – 619, August 2000.

[34]. X. Q. Gao, C. J. Duanmu and C. R. Zou, "A Mutlilevel Successive Elimination Algorithm for Block Matching Motion Estimation:, IEEE Transactions on Image Processing, Vol. 9, Issue 9, pp. 501 – 504, March 2000.

[35]. Soo-Mok Jung, Sung-Chul Shin, Hyunki Baik and Myong-Soon Park, “Efficient Multilevel Successive Eliminiation Algorithms for Block Matching Motion Estimation”, IEE Proceedings of Vision, Image and Signal Processing, Vol. 149, Issue 2, pp. 73 – 84, April 2002.

[36]. Tae Gyoung Ahn, Yong Ho Moon and Jae Ho Kim, “Fast Full-Search Motion Estimation Based on Multilevel Successive Elimination Algorithm”, IEEE Transactions on Circuits and Systems for Video Technology, Vol. 14, Issue 11, pp. 1265 – 1269, November 2004.

[37]. T. Koga, K. Linuma, A. Hirano, Y. Lijima and T. Ishiguro, “Motion Compensated Interframe Coding for Video Conferencing”, Proceedings of National Telecommunications Conference 81, pp. C.9.6.1 – 9.6.5, November 1981.

[38]. Reoxiang Li, Bing Zeng and M. L. Liou, “A New Three-Step Search Algorithm for Block Motion Estimation”, IEEE Transactions on Circuits and Systems for Video Technology, Vol. 4, Issue 4, pp. 438 – 442, August 1994.

[39]. Jo Yew Tham, Surendra Ranganath, Maitreya Ranganath, Ashraf Ali Kassim, “A Novel Unrestricted Center-Biased Diamond Search Algorithm For Block Motion Estimation”, IEEE Transactions on Circuits and Systems for Video Technology, Vol. 8, Issue 4, pp. 369 – 377, August 1998.

[40]. Ce Zhu, Xiao Lin and Lap-Pui Chau, “Hexagon-Based Search Pattern for Fast Block Motion Estimtion”, Vol. 12, Issue 5, pp. 349 – 355, May 2002.

[41]. Charles K. Chui, “An Introduction to Wavelets”, Academic Press, 1992.

[42]. Martin Vetterli, “Wavelets and Subband Coding”, Englewood Cliffs, NJ: Prentice Hall PTR, 1995.

[43]. Randy K. Young, “Wavelet Theory and Its Applications”, Kluwer Academic Publishers, 1993.

[44]. M. Antonini, M. Barlaud, P. Mathieu and I. Daubechies, “Image Coding Using Wavelet Transform”, IEEE Transactions on Image Processing, Vol. 1, No. 2, pp. 205 – 220, April 1992.

[45]. M. Vetterli and C. Herley, “Wavelets and Filter Banks: Theory and Design”, IEEE Transactions on Signal Processing, Vol. 40, Issue 9, pp. 2207 – 2232, September 1992.

[46]. Ingrid Daubechies, “Ten Lectures on Wavelets”, Philadelphia, Pa.: Society for Industrial and Applied Mathematics, 1992.

[47]. A. Skodras, C. Christopoulis and T. Ebrahimi, “The JPEG 2000 Still Image Compression Standard”, IEEE Signal Processing Magazine, Vol. 18, No. 5, pp. 36 – 58, September 2001.

[48]. Bryan E. Usevitch, “A Tutorial on Modern Lossy Wavelet Image Compression: Foundations of JPEG2000”, IEEE Signal Processing Magazine, pp. 22 – 35, September 2001.

[49]. Wim Sweldens, “The Lifting Scheme: A Construction of Second Generation Wavelets”, SIAM Journal on Mathematical Analysis archive, Vol. 29, Issue 2, pp. 511 – 546, March 1998.

[50]. Wim Sweldens, “The Lifting Scheme: A Custom-design Construction of Biorthogonal Wavelets”, Applied and Computational Harmonics Analysis, Vol. 3, No. 2, pp. 186 – 200, 1996.

[51]. Ingrid Daubechies and Wim Sweldens, “Factoring Wavelet Transforms into Lifting Steps”, J. Fourier Anal. Appl., Vol. 4, Nr. 3, pp. 247 – 269, 1998.

[52]. Andrew Secker and David Taubman, “Lifting-Based Invertible Motion Adaptive Transform (LIMAT) Framework for Highly Scalable Video Compression”, IEEE Transactions on Image Processing, vol. 12, no. 12, pp. 1530 – 1542, Dec 2003.

[53]. Nagita Mehrseresht and David Taubman, “Adaptively Weighted Update Steps in Motion Compensated Lifting Based Scalable Video Compression”, IEEE Proceedings of International Conference on Image Processing, vol. 2, pp. 771 – 774, Sept. 2003.

[54]. Christophe Tillier, Béatrice Pesquet-Popescu, Yinwei Zhan and Henk Heijmans, “Scalable Video Compression with Temporal Lifting using 5/3 Filters”, Proceedings of Picture Coding Symposium, pp. 55 – 58, April 2003.

[55]. Grégoire Pau and Béatrice Pesquet-Popescu, “Uniform Motion-Compensated 5/3 Filterbank for Subband Video Coding”, Proceedings of International Conference on Image Processing, vol. 5, pp. 3117 – 3120, Oct. 2004.

[56]. Deepak S. Turaga, Mihaela van der Schaar and Béatrice Pesquet-Popescu, “Temporal Prediction and Differential Coding of Motion Vectors in the MCTF Framework”, Proceedings of International Conference on Image Processing, vol. 2, pp. 57 – 60, Sept. 2003.

[57]. Jerome M. Shapiro, “Embedded Images Coding Using Zerotree Of Wavelet Coefficients”, IEEE Transactions on Signal Processing, Vol .41, No. 12, pp. 3445 – 3462, Dec. 1993.

[58]. Eui-Sung Kang, Toshihisa Tanaka, Tae-Hyung Lee and Sung-Jea Ko, “A Multi-threshold Embedded Zerotree Wavelet Coder”, Proceedings of International Technical Conference on Circuits / Systems, Computers and Communications (ITC – CSCC), Vol. 1, pp. 117 – 120, Jul. 1998.

[59]. Tanzeem Muzaffar and Tae-Sun Choi, “Simplified Wavelet Based Image Compression Using Fixed Length Residual Value”, Institute of Electronics, Information and Communication Engineers (IEICE) Transactions of Information and System, Vol. E84-D, No. 12, pp. 1828 – 1831, Dec. 2001.

[60]. Tanzeem Muzaffar and Tae-Sun Choi, “Wavelet Based Image Compression Using Subband Threshold”, SPIE International Symposium on Optical Science, Engineering and Instrumentation, Jul. 2002.

[61]. Stephen A. Martucci, Iraj Sodagar, Tihao Chiang and Ya-Qin Zhang, “A Zerotree Wavelet Video Coder”, IEEE Transactions on Circuits and Systems for Video Technology, Vol. 7, No. 1, Feb. 1997.

[62]. Carles D. Creusere, “A New Method of Roust Image Compression Based on the Embedded Zerotree Wavelet Algorithm”, IEEE Transactions on Image Processing, Vol. 6, No. 10, Oct. 1997.

[63]. S. Patel and S. Srinivasan, “Modified Embedded Zerotree Wavelet Algorithm for Fast Implementation of Wavelet Image Codec”, IEE Electronics Letters, Vol. 36, Issue 20, pp. 1713 – 1714, Sept. 2000.

[64]. Roberto Yusi Omaki, Gen Fujita, Takao Onoye and Isao Shirakawa, “Embedded Zerotree Wavelet Based Algorithm for Video Compression”, Proceedings of the IEEE Region 10 Conference (TENCON), Vol. 2, pp. 1343 – 1346, September 1999.

[65]. E. S. Kang, T. Tanaka and S. J. Ko, “Improved Embedded Zerotree Wavelet Coder”, Electronics Letters, Vol. 35, No. 9, pp. 705 - 706, April 1999.

[66]. Iraj Sodagar, Hung-Ju Lee, Paul Hatrack and Bing-Bing Chai, “Multi-Scale Zerotree Entropy Coding”, IEEE International Symposium on Circuits and Systems, Vol. 1, pp. 311 – 314, May 2000.

[67]. Il-Kyu Eom and Yoo-Shin Kim, “Multiple Description EZW Coding Using Overlapped Threshold”, Proceedings of International Conference on Signal Processing, Vol. 1, pp. 796 – 799, August 2002.

[68]. Yingwei Chen and William A. Pearlman, “Three-Dimensional Subband Coding of Video Using the Zerotree Method”, Proceedings of Symposium on Visual Communications and Image Processing, 1996.

[69]. Amir Said and William A. Pearlman, “Image Compression Using the Spatial-Orientated Tree”, Proceedings of International Symposium of Circuits and Systems, pp. 279 – 282, May 1993.

[70]. Amir Said and William A. Pearlman, “A New Fast and Efficient Image Codec Based on Set Partitioning Into Hierarchical Trees”, IEEE Transactions on Circuits and Systems for Video Technology, Vol. 6, pp. 243 – 250, Jun. 1996.

[71]. Beong-Jo Kim and William A. Pearlman, “An Embedded Wavelet Video Coder Using Three-Dimensional Set Partitioning in Hierarchical Trees (SPIHT)”, Proceedings of Data Compression Conference, pp. 251 – 260, March 1997.

[72]. Christos Chrysafis, Amir Said, Alex Drukarev, Asad Islam and William A. Pearlman, “SBHP – A Low Complexity Wavelet Coder”, IEEE International Conference on Acoustics, Speech and Signal Processing, Vol. 4, pp. 2035 – 2038, June 2000.

[73]. William A. Pearlman, Asad Islam, Nithin Nagaraj and Amir Said, “Efficient, Low-Complexity Image Coding With a Set-Partitioning Embedded Block Coder”, IEEE Transactions on Circuits and Systems for Video Technology, Vol. 14, No. 11, pp. 1219 – 1235, November 2004.

[74]. Ya-Qin Zhang and Sohail Zafar, “Motion-Compensated Wavelet Transform Coding for Color Video Compression”, IEEE Transactions on Circuits and Systems For Video Technology, Vol. 2, No. 3, pp. 285 – 296, September 1992.

[75]. M. K. Mandal, E. Chan, X. Wang and S. Panchanathan, “Multiresolution Motion Estimation Techniques for Video Compression”, Optical Engineering, Vol. 35, pp. 128 – 136, January 1996.

[76]. Jie Wei and Ze-Nian Li, “An Enhancement to MRMC Scheme in Video compression”, IEEE Transactions on Circuits and Systems for Video Technology”, Vol. 7, No. 3, pp. 564 – 568, June 1997.

[77]. Sohail Zafar, Ya-Qin Zhang and Bijan Jabbari, “Multiscale Video Representation Using Multiresolution Motion Compensated and Wavelet Decomposition”, IEEE Journal on Selected Areas in Communications, Vol. 11, No. 1, pp. 24 – 35, January 1993.

[78]. Seongman Kim, Seunghyeon Rhee, Jun Geun Jeon and Kyu Tae Park, “Interframe Coding Using Two-Stage Variable Block-Size Multiresolution Motion Estimation and Wavelet Decomposition”, IEEE Transactions on Circuits and Systems for Video Technology, Vol. 8, No. 4, pp. 399 – 410, August 1998.

[79]. Jungwoo Lee and Bradley W. Dickinson, “Subband Video Coding with Scene-Adaptive Hierarchical Motion Estimation”, IEEE Transactions on Circuits and Systems for Video Technology, Vol. 9, No. 3, pp. 459 – 466, April 1999.

[80]. Jinwen Zan, M. Omair Ahmad and M. N. S. Swamy, “New Techniques for Multi-resolution Motion Estimation”, IEEE Transactions on Circuits and Systems for Video Technology, Vol. 12, No. 9, pp. 793 – 802, September 2002.

[81]. Aria Nosratinia and Michael T. Orchard, “A Multi-resolution Framework for Backward Video Coding”, Proceedings of International Conference on Image Processing, pp. 563 – 566, 1995.

[82]. P. Cheng, J. Li and C. J. Kuo, “Multiscale Video Compression using Wavelet Transform and Motion Compensation”, Proceedings of IEEE International Conference on Image Processing, pp. 606 – 609, October 1995.

[83]. Hyun-Wook Park and Hyung-Sun 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.

[84]. Deepak S. Turaga, Mihaela van der Schaar and Béatrice Pesquet-Popescu, “Complexity Scalable Motion Compensated Wavelet Video Encoding”, IEEE Transactions on Circuits and Systems for Video Technology, Vol. 15, No. 8, pp. 982 – 993, August 2005.

[85]. Georgia Feideropoulou, Béatrice Pesquet-Popescu and Jean-Clude Belfiore, “Bit Allocation Algorithm for Joint Source-Channel Coding of T+2D Video Sequences”, Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing, Vol. 2, pp. 177 – 180, March 2005.

[86]. Grégoire Pau and Béatrice Pesquet-Popescu, “Four-Band Linear-Phase Orthogonal Spatial Filter Bank for Subband Video Coding”, Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing, Vol. 2, pp. 277 – 280, March 2005.

[87]. Bernd Girod and Sangeun Han, “Optimum Update for Motion-Compensated Lifting”, IEEE Signal Processing Letters, Vol. 12, No. 2, pp. 150 – 153, February 2005.

[88]. Sang-Hee Park, Hyo-Kak Kim, Jong-Wong Jung and Sung-Jea Ko, “Improved Motion Vector Prediction Method in Scalable Video Coding”, Proceedings of IEEE International Symposium on Intelligent Signal Processing and Communication Systems, pp. 253 – 256, December 2005.

[89]. Jens-Rainer Ohm, Mihaela van der Schaar and John W. Woods, “Interframe Wavelet Coding – Motion Picture Representation for Universal Scalability”, EURASIP Signal Processing: Image Communication, Special Issue on Digital Camera, Vol. 19, No. 9, pp. 877 – 908, October 2004.

[90]. Andrew Secker and David Taubman, “Highly Scalable Video Compression with Scalable Motion Coding”, IEEE Transactions on Image Processing, Vol. 13, No. 8, pp. 1029 - 1041, August 2004.

[91]. Jérôme Viéron, Christine Guillemot and Stéphane Pateux, “Motion Compensated 2D+T Wavelet Analysis for Low Rate FGS Video Compression”, Proceedings of Tyrrhenian International Workshop on Digital Communications, September 2002.

[92]. Cchristophe Parisot, Marc Antonini and Michel Barlaud, “3D Scan Based Wavelet Transform for Video Coding”, Proceedings of IEEE 4th Workshop on Multimedia Signal Processing, pp. 403 – 408, October 2001.

[93]. Janusz Konrad, “Transversal Versus Lifting Approach to Motion-Compensated Temporal Discrete Wavelet Transform of Image Sequences: Equivalence and Tradeoffs”, SPIE Symposium on Electronic Imaging, Visual Communications and Image Processing, pp. 452 – 453, January 2004.

[94]. Markus Flierl and Bernd Girod, “Investigation of Motion-Compensated Lifted Wavelet Transforms”, Proceedings of Picture Coding Symposium, April 2003.

[95]. Jizheng Xu, Shipeng Li, Zixiang Xiong and Ya-Qin Zhang, “Memory-Constrained 3D Wavelet Transforms for Video Coding without Boundary Effects”, IEEE Transactions on Circuits and Systems for Video Technology, Vol. 12, No. 9, pp. 812 – 818, September 2002.

[96]. Béatrice Pesquet-Popescu and Vincent Bottreau, “Three-Dimensional Lifting Schemes for Motion Compensated Video Compression”, Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing, Vol. 3, pp. 1793 – 1796, May 2001.

[97]. Vincent Bottreau, Marion Bénetière, Boris Felts, Béatrice Pesquet-Popescu, “A Fully Scalable 3D Subband Video Codec”, Proceedings of IEEE International Conference of Image Processing, Vol. 2, pp. 1017 – 1020, October 2001.

[98]. Deepak S. Turaga and Mihaela van der Schaar, “Reduced Complexity Spatio-Temporal Scalable Motion Compensated Wavelet Video Coding”, Proceedings of IEEE International Conference on Multimedia and Expo, Vol. 2, pp. 561 – 564, July 2003.

[99]. Christophe Tillier and Béatrice Pesquet-Popescu, “3D, 3-Band, 3-Tap Temporal Lifting for Scalable Video Coding”, Proceedings of IEEE International Conference on Image Processing, Vol. 3, pp. 779 – 782, September 2003.

[100]. Christophe Tillier, Béatrice Pesquet-Popescu and Mihaela van der Schaar, “Highly Scalable Video Coding by Bidirectional Predict-Update 3-Band Schemes”, Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing, Vol. 3, pp. 125 – 128, May 2004.

[101]. Markus Flierl, “Video Coding with Lifted Wavelet Transforms and Frame-Adaptive Motion Compensatoin”, Proceedings of the 8th International Workshop on Very Low Bitrate Video Coding, September 2003.

[102]. Mihaela van deer Schaar and Deepak S. Turaga, “Unconstrained Motion Compensated Temporal Filtering (UMCTF) Framework for Wavelet Video Coding”, Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing, Vol. 3, pp. 81 – 84, April 2003.

[103]. Grégoire Pau, Christophe Tiller and Béatrice Pesquet-Popescu, “Optimization of the Predict Operator in Lifting-Based Motion Compensated Temporal Filtering”, Proceedings of SPIE Symposium on Visual Communications and Image Processing, January 2004.

[104]. Thomas André, Marco Cagnazzo, Marc Antonini, Michel Barlaud, Nikola Božinvoić and Janusz Konrad, “(N, 0) Motion-Compensated Lifting-Based Wavelet Transform”, Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing, Vol. 3, pp. 121 – 124, May 2004.

[105]. Christophe Tillier, Béatrice Pesquet-Popescu and Mihaela van der Schaar, “Weighted Average Spatio-Temporal Update Operator for Subband Video Coding”, Proceedings of IEEE International Conference on Image Processing, Vol. 2, pp. 1305 – 1308, October 2004.

[106]. Konstantin Hanke, Jens-Rainer Ohm and Thomas Rusert, “Adaptation of Filters and Quantization in Spatio-Temporal Wavelet Coding with Motion Compensation”, Proceedings of Picture Coding Symposium, pp. 49 – 54, April 2003.

[107]. William A. Pearlman, Asad Islam, Nithin Nagaraj and Amir Said, “Efficient, Low-Complexity Image Coding with a Set-Partitioning Embedded Block Coder”, IEEE Transactions on Circuits and Systems For Video Technology, pp. 1219 – 1235, Vol. 14, No. 11, Nov. 2004.

[108]. K. C. Hui, W. C. Siu and Y. L. Chan, “New Adaptive Partial Distortion Search Using Clustered Pixel Matching Error Characteristic”, ISCAS, pp. 97-100, 2004.

[109]. B. J. Kim, Z. Xiong and W. A. Pearlman, “Very low bit-rate embedded video coding with 3D set partitioning in hierarchical trees (3D-SPIHT)”, IEEE Transactions on Circuits and Systems for Video Technology, vol. 8, pp. 1365 – 1374, 2003.

[110]. S. Hsiang and J. Woods, “Embedded image coding using zeroblocks of subband/wavelet coefficients and context modelling”, IEEE International Symposium on Circuits and Systems, p. 589, 2000.