DCT VIDEO COMPOSITING WITH EMBEDDED ZEROTREE CODING FOR MULTI-POINT VIDEO CONFERENCING

by

Hakkı Alparslan Ilgın

BS, Ankara University, Ankara, Turkey, 1993

MS, Ankara University, Ankara, Turkey, 1997

Submitted to the Graduate Faculty of

the School of Engineering in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2004

UNIVERSITY OF PITTSBURGH

SCHOOL OF ENGINEERING

This dissertation was presented

by

Hakkı Alparslan Ilgın

It was defended on

December 3, 2004

and approved by

Dr. Luis F. Chaparro, Professor, Department of Electrical and Computer Engineering

Dr. Ching-Chung Li, Professor, Department of Electrical and Computer Engineering

Dr. J. Robert Boston, Professor, Department of Electrical and Computer Engineering

Dr. Heung-no Lee, Professor, Department of Electrical and Computer Engineering

Dr. Michael McCloud, Professor, Department of Electrical and Computer Engineering

Dr. Juan J. Manfredi, Professor, Department of Mathematics

Thesis Director: Dr. Luis F. Chaparro, Professor, Department of Electrical and Computer Engineering

ABSTRACT

DCT VIDEO COMPOSITING WITH EMBEDDED ZEROTREE CODING FOR MULTI-POINT VIDEO CONFERENCING

Hakkı Alparslan Ilgın, PhD

University of Pittsburgh, 2004

In this thesis, DCT domain video compositing with DCT-based embedded zerotree coding for multi-point video conferencing is considered. In a typical video compositing system, video sequences coming from different sources are composited into one video stream and sent using a single channel to the receiver points. There are mainly three stages of video compositing: decoding of incoming video streams, decimation of video frames, and encoding of the composited video. Conventional spatial domain video compositing requires transformations between the DCT and the spatial domains increasing the complexity of computations. The advantage of the DCT domain video compositing is that the decoding, decimation and encoding remain fully in the DCT domain resulting in faster processing time and better quality of the composited videos. The composited videos are encoded via a DCT-based embedded zerotree coder which was originally developed for wavelet coding. An adaptive arithmetic coder is used to encode the symbols obtained from the DCT-based zerotree coding resulting in embedded bit stream. By using the embedded zerotree coder the quality of the composited videos is improved when compared to a conventional encoder. An advanced version of zerotree coder is also used to increase the performance of the compositing system. Another improvement is due to the use of local cosine transform to decrease the blocking effect at low bit rates. We also apply the proposed DCT decimation/interpolation for single stream video coding achieving better quality than regular encoding process at low bit rates. The bit rate control problem is easily solved by taking the advantage the embedded property of

zerotree coding since the coding control parameter is the bit rate itself. We also achieve the optimum bit rate allocation among the composited frames in a GOP without using subframe layer bit rate allocation, since zerotree coding uses successive approximation quantization allowing DCT coefficients to be encoded in descending significance order.

Keywords: Video coding, multi-point video conferencing, video compositing, DCT transcoding, DCT decimation/interpolation, image resizing, DCT block transformation, motion estimation, motion compensation, embedded zerotree coding, significance tree coding, hierarchical image coding, set partitioning in hierarchical trees, successive approximation quantization, adaptive arithmetic coding, bit rate control.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

DCT	Discrete Cosine Transform
LCT	Local Cosine Transform
MC	Motion Compensation
ME	Motion Estimation
Q	Quantization
QP	Quantization Parameter
EZT	Embedded Zerotree Coding
DCT-EZT	DCT-Based Embedded Zerotree Coding
EZW	Embedded Zerotree Wavelet Coding
SAQ	Successive Approximation Quantization
PSNR	Peak Signal to Noise Ratio
MSE	Mean Square Error
MAD	Mean Absolute Difference
CIF	Common Intermediate Format
QCIF	Quadrature Common Intermediate Format
JPEG	Joint Photographic Experts Group
MPEG	Moving Picture Experts Group
GOP	Group of Pictures
LOT	Lapped Orthogonal Transform

ACKNOWLEDGEMENT

I would like to thank my advisor Dr. Luis F. Chaparro for his guidance, helps and supports. I also would like to thank my friends and colleagues Abdullah A. Al-Shehri, Jongchin *"James"* Chien, Xiaoping Hu, and Jian Xu. Special thanks to Dr. C. C. Li.

Finally, I want to thank my dear wife Hülya Ilgın for always being with me, and my dear mother Fatma Ilgın for always being there for me, and for their great support, encouragement and patience during my Ph. D. study. I dedicate this thesis to my wife and my mother. I am also thankful to the other members of my family; my sister Dilek Hale Şahin, my brother Muhammet Ilgın, and my grandmother Ahsene Yalnız.

1.0 INTRODUCTION

Video compression techniques are becoming more advanced and widespread with the increasing demand and popularity of video applications. Since bandwidth or storage media constrain the data to be transmitted or stored, video compression methods are being improved to reduce the amount of information while providing a higher video quality. Besides high compression ratio, especially in real-time video applications such as video conferencing, fast processing of a video sequence without losing much of its quality is required.

The basic principle of video compression is to minimize the redundancies in the video sequence. These redundancies are spatial redundancy present in a video frame, temporal redundancy or the similarities between two successive frames, and the redundancy between the compressed data symbols. The spatial redundancy among pixels is reduced by employing intraframe compression. Transform coding techniques such as the Discrete Cosine Transform (DCT), which is commonly used in most of the standard video codecs, reduces the correlation among the pixels in a video frame. The advantage of using transform coding is that most of the energy is mainly concentrated in a few low frequency transform coefficients making the other coefficients less significant. After obtaining transform coefficients, compression is achieved by using methods such as scalar quantization, vector quantization, embedded zerotree coding or any other lossy compression technique.

Temporal redundancy is reduced by interframe coding. A predictive coder decreases the temporal difference between two consecutive frames with help of motion estimation (ME) and motion compensation (MC). An ME algorithm basically involves displacement measurement and error calculation. The most complicated part of the algorithm is the displacement measurement procedure, which searches for the optimal reference block in the previous frame. MC algorithms simply re-obtain the optimal block by using the motion

vectors estimated by the ME algorithm. The quantized transform coefficients are coded by an entropy coder.

The encoder and the decoder of a basic video coding system is shown in Fig. 1 and 2, respectively. This hybrid codec implements compression of video sequences in both spatial and DCT domains. At the encoder, the incoming video frame is subtracted from the motion compensated previous frame in the spatial domain. The obtained error frame, e, is transformed into the DCT domain and quantized. The quantized error frame, E_q, and motion vectors, MV, are then entropy-encoded and sent to the decoder. The quantized error frame is also inverse-quantized and transformed back to the spatial domain to be added to the previous frame to prevent the errors from being cumulative. The feedback structure is identical in the decoder except the motion estimation part that does not exist in the decoder.

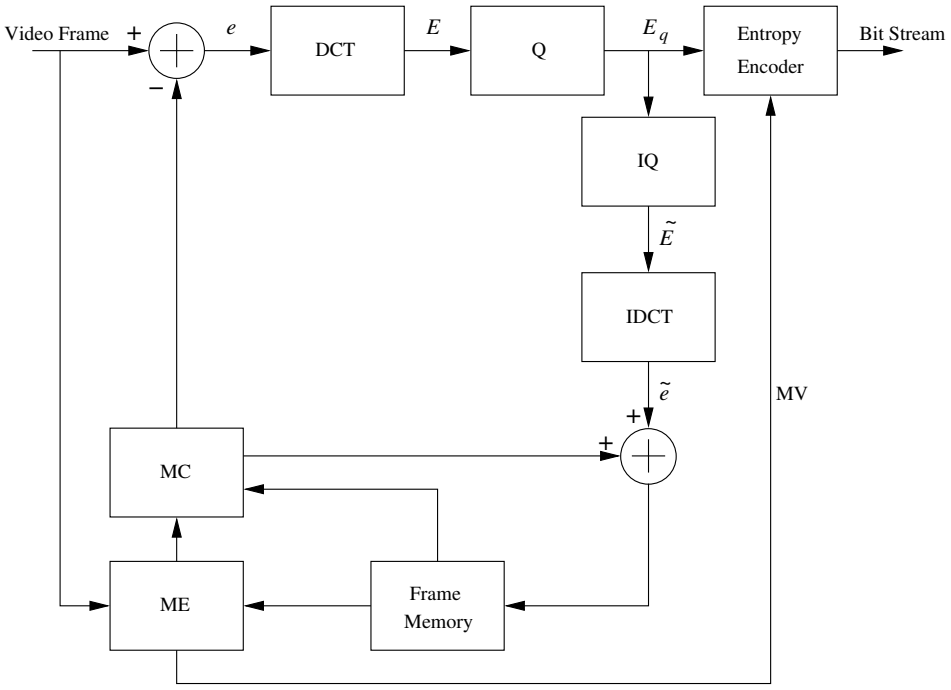


Figure 1: Hybrid encoder

A conventional hybrid decoder has an entropy decoder that converts bit streams back to the quantized DCT error frame and motion vectors. The previous frame in the frame memory is motion-compensated by using the motion vectors. The current frame is reconstructed by

adding the inverse-quantized error frame to the motion-compensated previous frame in the spatial domain.

For video conferencing applications, processing speed is an important factor for real-time communication. Typically, video conferencing standards such as H.263 and MPEG, are widely used for low bit rate transmission. But, work on increasing the quality of video while achieving more compression continues. Furthermore, in real-time applications requiring additional processings, such as multi-point video conferencing, the significance of the processing speed becomes more important.

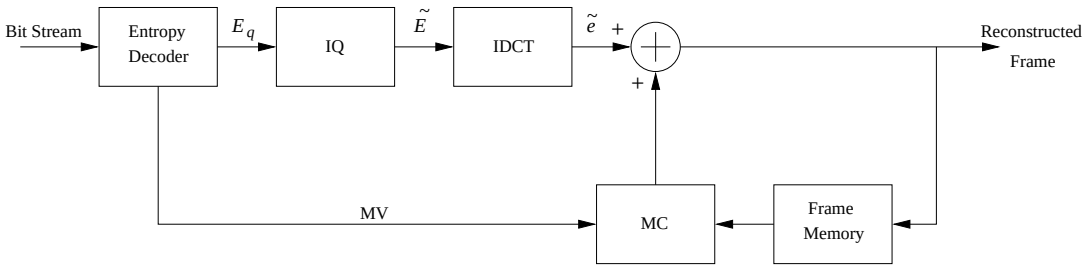


Figure 2: Hybrid decoder

In a typical multi-point video conferencing system [3], the videos coming from different sources are put together (composited) and sent using a single channel. To composite the incoming videos the first step is to decode them by using hybrid decoders. The problem here is the transformation between the DCT and spatial domains which requires high computational complexity, also possible aliasing is caused by the decimation process in the spatial domain needed to composite the videos. The composited videos are re-encoded by a hybrid encoder requiring another transformation from spatial to DCT domain. Finally, bit rate control of the encoded bit stream is another issue to bear in mind.

The work presented in thesis concentrates on developing fast compositing fully in the DCT domain with DCT-based embedded zerotree coding for real-time multi-point video conferencing. We propose a new decimation/interpolation method to resize video frames in the DCT domain for integer and rational decimation factors. We also use embedded zerotree

coding method to code the composited DCT error frames. Beside especially wavelet based zerotree coding is commonly used for image coding, we implement this method to simplify the bit rate control problem and improve the quality of the composited videos. We take advantage of the zerotree coding method to control the bit rate and optimally distribute the bit budget among the frames and the subframes of composited video sequences. We also introduce the proposed DCT decimation/interpolation method for single stream video coding achieving better results than regular coding at low bit rates.

The rest of the material is organized as follows. In the next chapter, multi-point video conferencing, advantages of compositing in the DCT domain over that in the spatial domain, and DCT transcoding are explained. DCT domain decimation is also given in this chapter. In the same chapter, we illustrate DCT decimation algorithms which have less computational complexity and obtain better results when compared to other DCT decimation algorithms. In the third chapter, embedded zerotree coding of DCT coefficients in a rearranged structure is presented. Features of the DCT embedded zerotree coding and the details of successive approximation quantization are also included in the third chapter. Then a simple example of embedded zerotree coding is given. Experimental results are also given in details in this chapter. We show that the DCT-based embedded zerotree coding gives better results than conventional DCT domain coding in terms of video quality and compression ratio. In the last section of Chapter 3, we apply the proposed DCT decimation/interpolation and embedded zerotree coding to single stream video coding at low bit rates achieving better results compared to conventional coders. In Chapter 4, we investigate an improved version of zerotree coding called set partitioning in hierarchical trees to improve the coding efficiency. We also compare the two zerotree coding methods in the same chapter. In the second section of Chapter 4, we use local cosine transform (LCT) with the proposed codec to reduce the blocking effect at low bit rates. Bit rate control problem is considered in Chapter 5. We use convex rate-distortion model to achieve optimized bit rates allocated to each frame in a group of frames (GOP). Comparison of convex and piecewise linear rate-distortion models are given in the same chapter. The last chapter consists of conclusions and possible future work.

2.0 VIDEO COMPOSITING FOR MULTI-POINT VIDEO CONFERENCING

Video compositing for multi-point video conferencing includes decoding, decimation and re-encoding processes. Video streams received from different locations are composited into one video stream typically to save bandwidth. Compositing can be done within the network or at a final receiver point. To transmit the composited video to each viewer site, thus saving communication bandwidth and reducing connection overhead, compositing must be done within the network [3].

The most computationally intensive parts of the whole process are inverse and forward DCT computations, and re-implementation of motion estimation and compensation. Therefore, the best approach for video compositing is to directly composite videos in the compressed domain instead of the spatial domain. This "compressed input-compressed output" approach saves calculations by avoiding the complexity of inverse and forward DCT operations [1, 3, 4, 5, 6].

In a spatial domain video compositing system, the incoming video streams are decoded by conventional decoders such as H.263 decoders (see Fig. 3). After decoding the bit stream by an entropy decoder, DCT errors, $\{E_i\}$, and motion vectors, $\{MV_i\}$ are obtained. An H.263 decoder transforms video error frames from DCT domain to spatial domain. Accordingly, motion compensation is done in the spatial domain. Then video sequences are decimated and composited. The composited video sequence needs to be re-encoded before being transmitted to the users. This is basically performed by an H.263 encoder. The encoded video sequence is again in the DCT domain. In Fig. 3, compositing of four different video sequences is displayed. As will be seen later, compositing of different number of incoming video streams is possible.

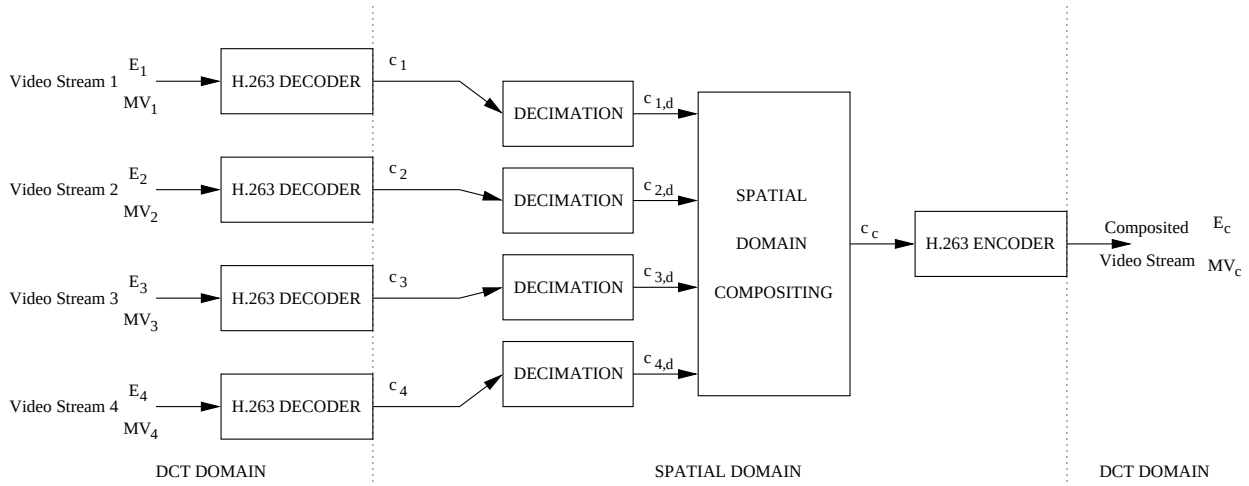


Figure 3: Spatial domain compositing

The other approach for video compositing is to compose the videos fully in the transform domain as shown in Fig. 4. This method does not require transformations between spatial and DCT domains. Consequently a computational speedup occurs. The incoming video streams are decoded by DCT transcoders directly into JPEG-type of images in the DCT domain. Then DCT decimation and compositing are performed. Finally, the composited video is compressed by performing motion estimation and compensation in the DCT domain.

The DCT transcoding procedure decodes the incoming video by utilizing the motion compensation without converting it back into the spatial domain (see Fig. 5). Differently from a conventional hybrid decoder, the DCT approach does not allow to view the decoded videos. However, more importantly it has the advantage of avoiding time delays that occur during the inverse DCT transformation. Next the basic principles of the DCT transcoding and DCT motion compensation are explained. We also show the implementation of a fast transcoder that we used in this work.

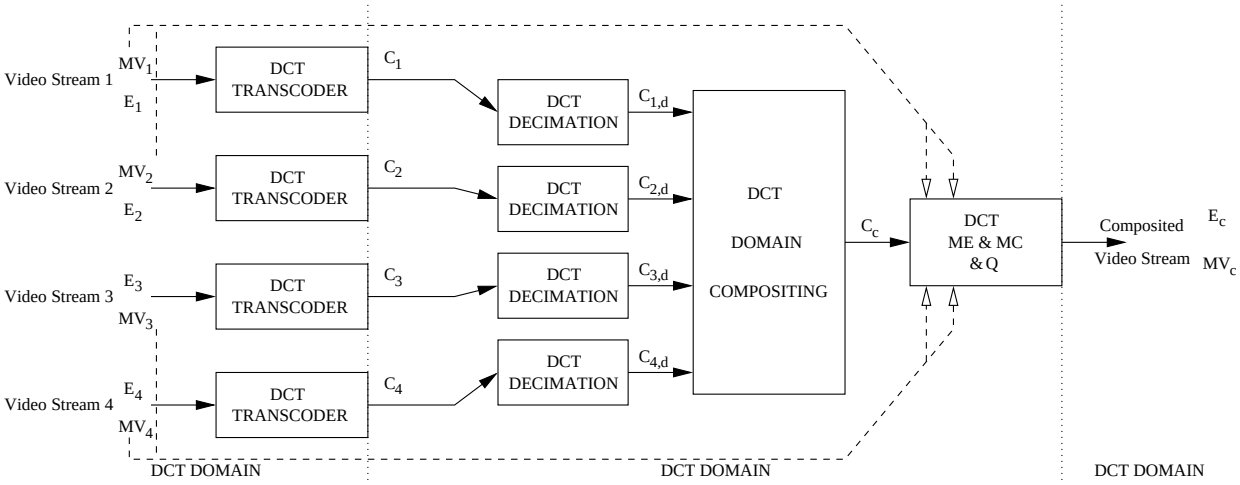


Figure 4: DCT domain compositing

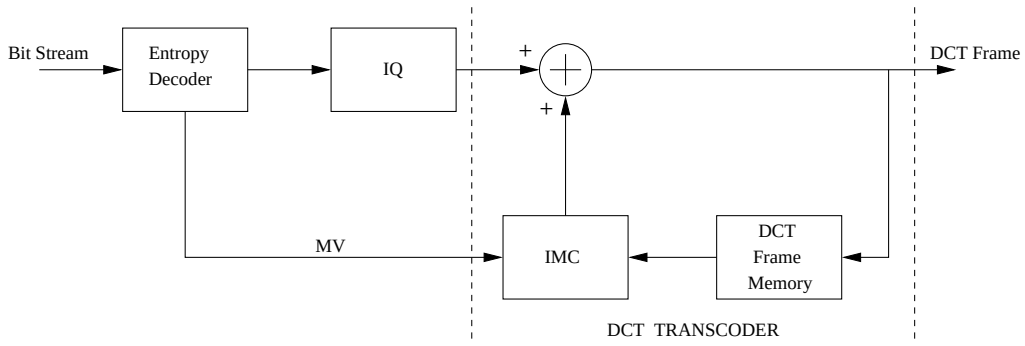


Figure 5: DCT transcoder

2.1 MOTION COMPENSATION IN THE DCT DOMAIN

As shown, a DCT transcoder has the same structure as a hybrid decoder except it does motion compensation in the DCT domain. Before explaining the motion compensation first consider the motion estimation in the spatial domain. Several criteria like Mean Square Error (MSE), Mean Absolute Difference (MAD), and Sum Absolute Difference (SAD) are commonly used to obtain the best matched, or namely, the optimal block for the current block [27, 29]. For instance, if MAD is used for an $M \times M$ block size the following error

$$MAD(i,j) = \frac{1}{M^2} \sum_{k=0}^{M-1} \sum_{l=0}^{M-1} |c_t(x+k, y+l) - c_{t-1}(x+k+i, y+l+j)| \tag{2.1}$$

is computed between the $c_t(x+k, y+l)$ that stands for the pixels of the block in the current frame at time t, and the $c_{t-1}(x+k+i, y+l+j)$ that corresponds to the pixels of the block in the previous (reference) frame. Here (x, y) and $(x+i, y+j)$ are the spatial coordinates of the top left corner of the blocks $c_t(x, y)$ and $c_{t-1}(x+i, y+j)$ respectively. The motion vectors are defined as $-M \leq i \leq M$ horizontally, and $-M \leq j \leq M$ vertically. If $(x+p, y+q)$ is found to be the coordinates of the optimal block $c_{t-1}(x+i, y+i)$ for which $MAD(i,j)$ is minimized, hence the motion vector for the block $c_t(x, y)$ is $(i, j) = (p, q)$.

If motion estimation is performed in the DCT domain, the MAD criterion is written as

$$MAD_{DCT}(i,j) = \frac{1}{M^2} \sum_{k=0}^{M-1} \sum_{l=0}^{M-1} |C_t(x+k, y+l) - C_{t-1}(x+k+i, y+l+j)| \tag{2.2}$$

where $C_t(x+k, y+l)$, and $C_{t-1}(x+k+i, y+l+j)$ are the DCT values of the blocks in the current, and the previous frames respectively.

Now let us consider the spatial domain motion compensation. Basically, in the previous frame when adding the motion vector to the coordinates of the current block, the optimal block is located (see Fig. 6). Hence by adding the block in the error frame, $e_t(x, y)$, to the optimal block in the motion-compensated frame, $\hat{c}_{t-1}(x+p, x+q)$, we obtain the current block

$$c_t(x, y) = e_t(x, y) + \hat{c}_{t-1}(x+p, y+q) \tag{2.3}$$

Eq. (2.3) also can be written in DCT domain as

$$C_t(x,y) = E_t(x,y) + \hat{C}_{t-1}(x+p, y+q) \tag{2.4}$$

where $C_t(x,y) = \text{DCT}\{c_t(x,y)\}$, $E_t(x,y) = \text{DCT}\{e_t(x,y)\}$, and $\hat{C}_{t-1}(x+p,y+q) = \text{DCT}\{\hat{c}_{t-1}(x+p,y+q)\}$. So, the basic idea of motion compensation in the DCT domain is similar to that in the spatial domain. However, the optimal DCT block $\hat{C}_{t-1}(x+p,y+q)$ may not correspond to the DCT block of the previous frame. Therefore it needs to be calculated by using the DCT blocks in the neighboring area. As displayed in the Fig. 6, for four DCT blocks, R_1, R_2, R_3, and R_4, the optimal DCT block is computed by using the covered parts of each four blocks.

Let us first consider the spatial domain approach for combining those four blocks

$$\hat{c} = \sum_{i=1}^{4} v_i r_i h_i \tag{2.5}$$

where v_i, and h_i are sparse matrices that are used to window and shift the matrices r_i vertically, and horizontally [5, 7]. For simplicity, the optimal block, $\hat{c}_{t-1}(x+p,y+q)$ is shown as $\hat{c}$.

According to the definition of the DCT and its orthonormality, Eq. (2.5) becomes

$$\hat{C} = \sum_{i=1}^{4} V_i R_i H_i \tag{2.6}$$

where $V_i = \text{DCT}\{v_i\}$, $R_i = \text{DCT}\{r_i\}$, and $H_i = \text{DCT}\{h_i\}$. From the definition and considering $M = 8$, the two-dimensional DCT of an 8×8 block is

$$R_i = \text{DCT}\{r_i\} = S^8 r_i (S^8)^t \tag{2.7}$$

where S^8 is 8×8 DCT operation matrix. Using the orthonormality property of S^8

$$(S^8)^t S^8 = I, \tag{2.8}$$

we then have from Eq. (2.5)

$$S^8 \hat{c} (S^8)^t = \sum_{i=1}^{4} S^8 v_i (S^8)^t S^8 r_i (S^8)^t S^8 h_i (S^8)^t \tag{2.9}$$

which equals Eq. (2.6). For example, in the spatial domain, the upper left part of the optimal block, $\tilde{r}_1$, in Fig. 7 is computed as

$$\tilde{r}_1 = \begin{bmatrix} \mathbf{0} & I^a \\ \mathbf{0} & \mathbf{0} \end{bmatrix} r_1 \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ I^w & \mathbf{0} \end{bmatrix} \tag{2.10}$$

where the matrix on the left, which is v_1, provides vertical windowing and shifting by a, and the matrix on the right, which is h_1, makes a horizontal windowing and shifting by w. The DCT domain windowing and shifting is obtained by simply using the DCTs of these matrices as follows

$$\tilde{R}_1 = V_1 R_1 H_1.$$

In Eq. (2.10), I^a and I^w are identity matrices of dimension a and w. The height a, and the width w are easily obtained from the motion vector (p, q), and the block size M. For example, in the Fig. 7, a is equal to $|q|$, and w is equal to $M - p$. The DCT domain motion compensation implies storage of V_i, and H_i matrices for all possible motion vectors. However only $2M - 2$ matrices need to be stored because of the similarities of the matrices V_i, and H_i [7]. Furthermore, this method is faster than the method that computes the inverse DCT of each four block, and computes back to the DCT of the optimal block [3].

Even though the matrices v_i and h_i are sparse, DCTs of these matrices, V_i and H_i, are not sparse. Thus computational complexity of Eq. (2.6) is more than that of Eq. (2.5). However, in most cases, this does not keep the DCT transcoder from being faster than the hybrid decoder since there is no inverse DCT computation in the DCT transcoder. Furthermore, the position of the optimal block is a factor which affects the decoding time. The optimal DCT block may be a combination of two DCT blocks instead of four. The other possibility is that optimal DCT block may be one of the DCT blocks in the search area that does not require any computation. Possible positions of the optimal DCT block are shown in Fig. 8. A running time comparison of DCT transcoder versus hybrid decoder is displayed in Fig. 9 for four CIF (Common Intermediate Format) video sequences of size 288×352. In the next subsection we will introduce a faster transcoder which uses sparse matrices for windowing and shifting.

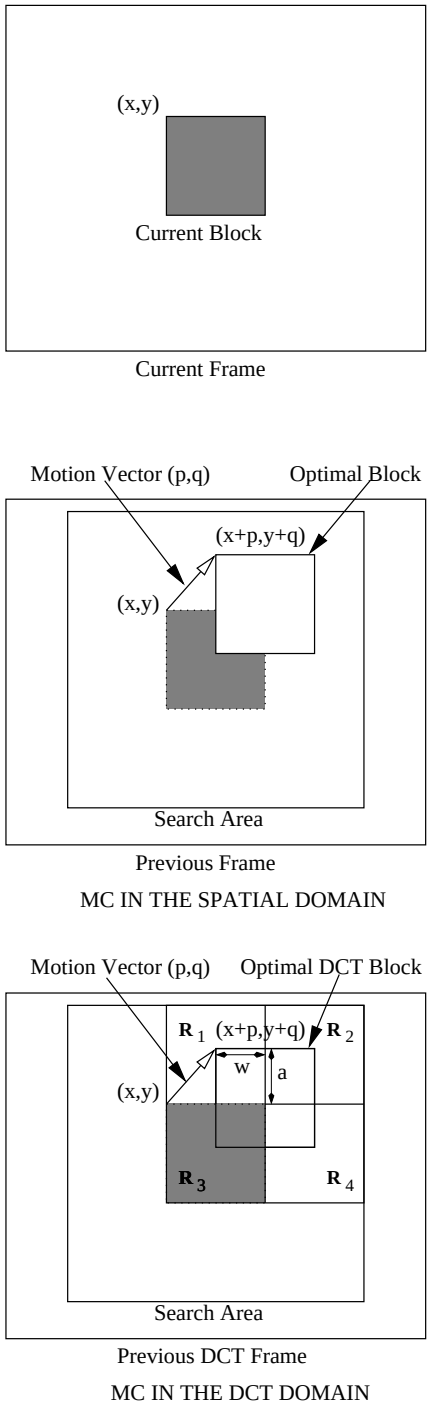


Figure 6: Motion compensation in the spatial and the DCT domains

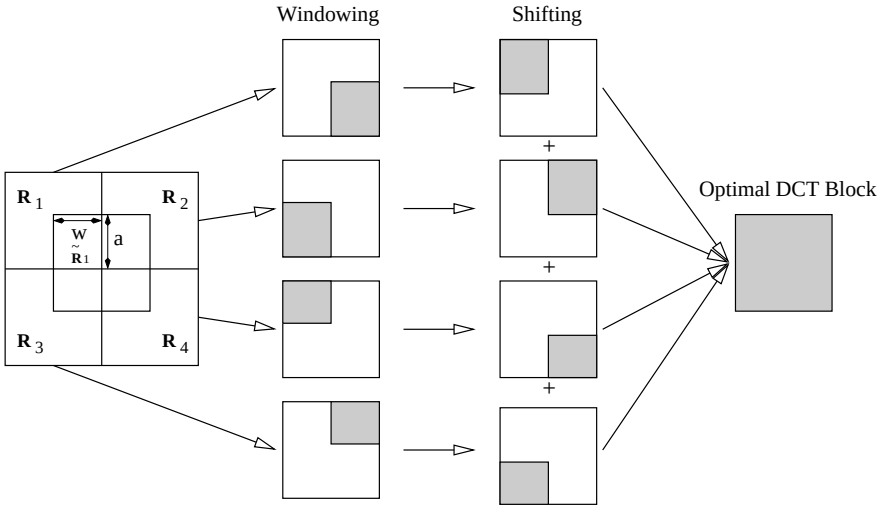


Figure 7: Windowing and shifting

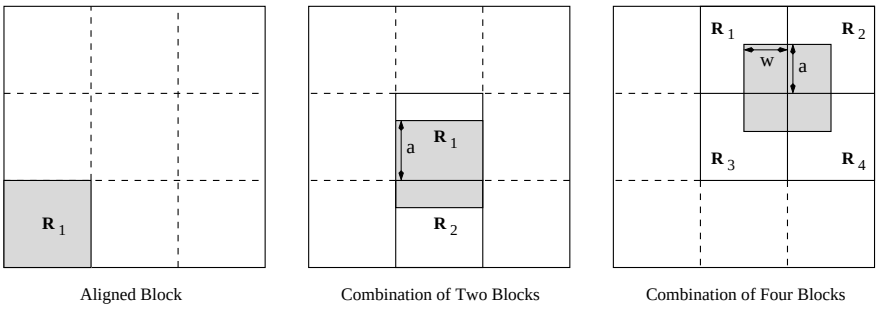


Figure 8: Possible positions of optimal DCT block

2.1.1 Fast DCT Transcoding

To improve the transcoder and to overcome the latency problem caused by computing the nonaligned blocks, sparse matrices can be used instead of the nonsparse matrices V_i, and H_i in Eq. (2.6) [9, 10, 11]. By using the similarities among these matrices and considering that the block size is 8×8, which is the block size in video coding standards such as H.26x and MPEG, Eq. (2.6) is rewritten as

$$\hat{C} = F_a R_1 G_w + F_a R_2 F_{8-w} + G_{8-a} R_3 G_w + G_{8-a} R_4 F_{8-w} \tag{2.11}$$

where $F_a = V_1 = V_2$, $G_{8-a} = V_3 = V_4$, $G_w = H_1 = H_3$, and $F_{8-w} = H_2 = H_4$. Hence, by using the definition of the DCT of a matrix in Eq. (2.7), Eq. (2.11) becomes

$$\hat{C} = S^8[f_a(S^8)^t(R_1 S^8 g_w + R_2 S^8 f_{8-w}) + g_{8-a}(S^8)^t(R_3 S^8 g_w + R_4 S^8 f_{8-w})](S^8)^t \tag{2.12}$$

or

$$\hat{C} = S^8[(f_a(S^8)^t R_1 + g_{8-a}(S^8)^t R_3) S^8 g_w + (f_a(S^8)^t R_2 + g_{8-a}(S^8)^t R_4) S^8 f_{8-w}](S^8)^t \tag{2.13}$$

The 8×8 DCT operation matrix S^8 can be written as products of sparse matrices as follows [9]

$$S^8 = DPB_1 B_2 M_1 A_1 A_2 A_3 \tag{2.14}$$

where D is a diagonal matrix, and P, B_1, B_2, M_1, A_1, A_2 and A_3 are sparse matrices, which their entries are given in Appendix A.

By defining pre-computed matrices

$$k_i = f_i(M_1 A_1 A_2 A_3)^t,\ 1 \leq i \leq 8$$

and

$$l_i = g_i(M_1 A_1 A_2 A_3)^t,\ 1 \leq i \leq 8$$

Table 1: Average speed improvements by DCT transcoder and fast DCT transcoder over hybrid decoder

	Average Speed Improvements (%)	
	DCT Transcoder	Fast DCT Transcoder
Trevor	25.4	43.3
Claire	22.8	47.6
Salesman	33.3	43.3
Hall	33.7	43.1

and by using the factorization of S_8 in Eq. (2.14), Equations (2.12) and (2.13) are respectively obtained as

$$\hat{C} = S^8[k_a B_2^t B_1^t P^t D(R_1 DPB_1 B_2 k_w^t + R_2 DPB_1 B_2 l_{8-w}^t) \\ + l_{8-a} B_2^t B_1^t P^t D(R_3 DPB_1 B_2 k_w^t + R_4 DPB_1 B_2 l_{8-w}^t)](S^8)^t \tag{2.15}$$

and

$$\hat{C} = S^8[(k_a B_2^t B_1^t P^t DR_1 + l_{8-a} B_2^t B_1^t P^t DR_3) DPB_1 B_2 k_w^t \\ + (k_a B_2^t B_1^t P_t DR_2 + l_{8-a} B_2^t B_1^t P^t DR_4) DPB_1 B_2 l_{8-w}^t](S^8)^t \tag{2.16}$$

The optimal block is computed by using either Eq. (2.15) or (2.16) depending on which one requires less computations for the given a and w. The running time comparisons of the hybrid decoder, DCT transcoder and the fast DCT transcoder which includes sparse matrices are shown in Fig. 9. As shown in the figure, by using the sparce matrices DCT transcoder becomes faster. On average, the improvements of the speed by the DCT transcoder and the fast DCT transcoder are about 28.8 % and 44.3 % respectively. Average speed improvements for each video sequence are shown in Table 1.

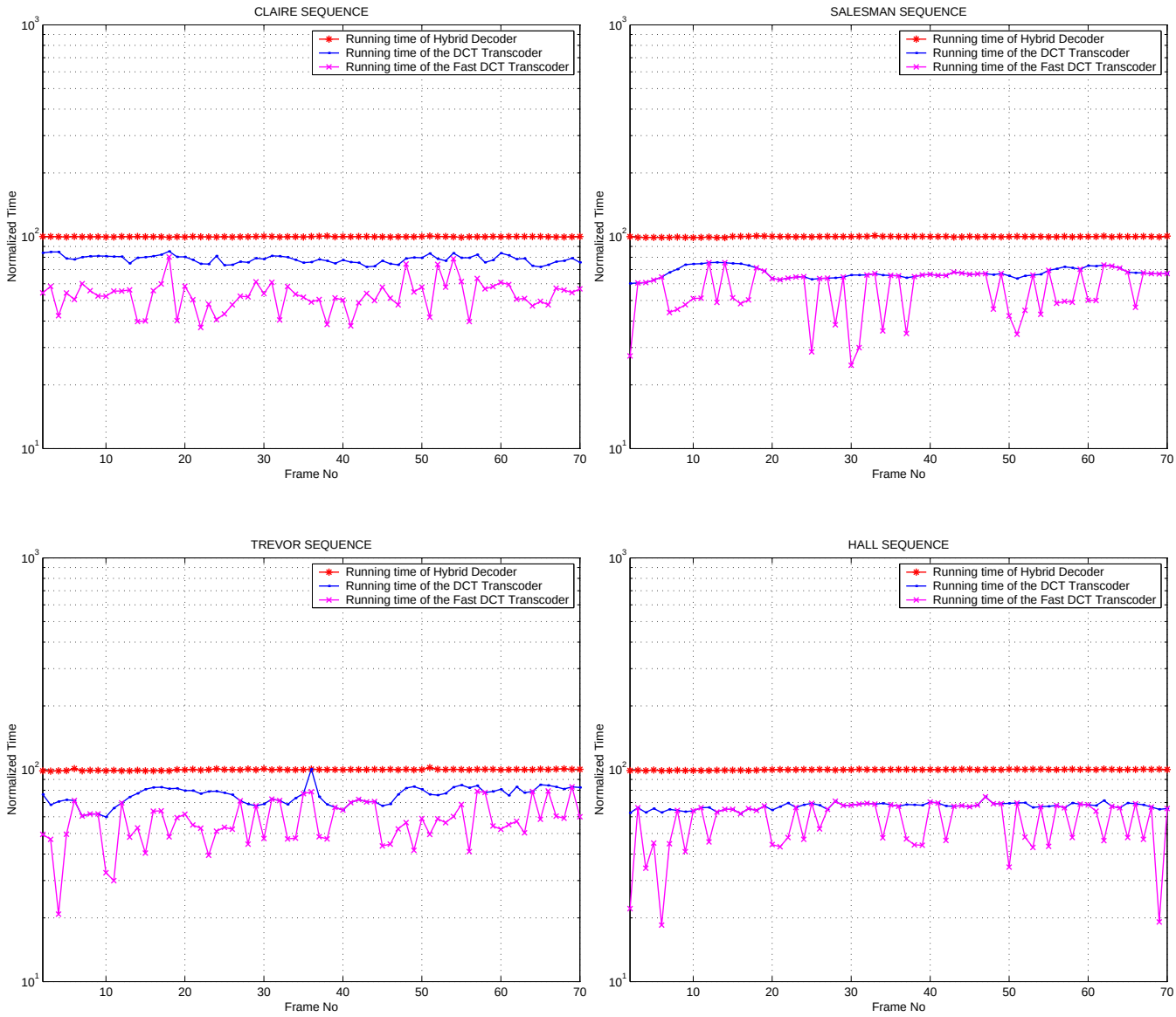


Figure 9: Running time comparisons of hybrid decoder, DCT transcoder and fast DCT transcoder

2.2 DECIMATION IN THE DCT DOMAIN

As in the other stages of multi-point videoconferencing, like decoding incoming video sequences in the spatial domain, straightforward techniques for spatial domain scaling of compressed video via inverse DCT transformation and re-transformation are computationally expensive [8]. Therefore, recently there has been great efforts to develop fast DCT domain decimation techniques [9, 10, 12, 14]. In this section, we consider fast algorithms for DCT decimation. To decimate by an integer factor, N, an array of $N \times N$ DCT blocks, whose sizes are 8×8, is first transformed into an $8N \times 8N$ DCT block and then masked to obtain the low frequency coefficients that gives the decimated 8×8 DCT block. The process is shown in Fig. 10 for $N = 2$. Our algorithm can also be used for rational values of N, such as $N = 2/3$, or $N = 3/4$. In this case, some additional computations are needed as will be explained later in this section.

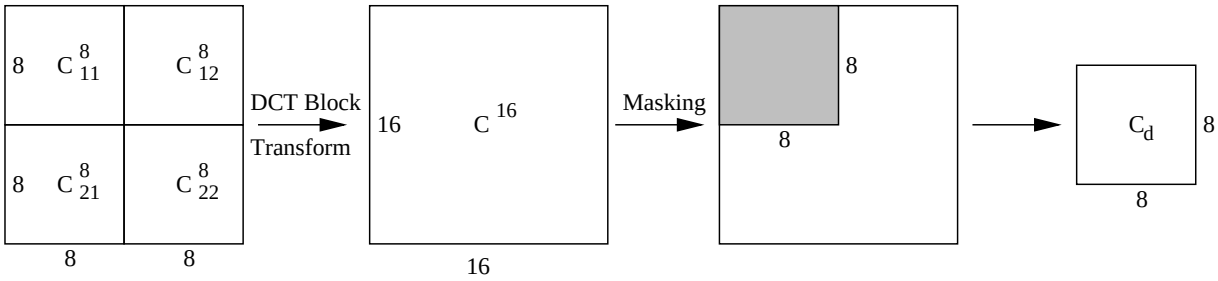


Figure 10: DCT decimation process for $N = 2$

The structure of the composited video depends on the decimation factor N [26]. For example, when $N = 2$ there is a total of four subframes in the composited video. When $N = 3$, the composited video will include a total of nine different video frames in 3×3 matrix form. For the rational case, for instance when $N = 2/3$, six video sequences are composited into one, five videos being decimated by 3, and the other decimated by 2/3. Several compositing structures are shown in Fig. 11.

In the spatial domain, decimation is done by low-pass filtering and discarding every other row and column of the video frames [1, 2]. However spatial domain decimation may cause

aliasing resulting in image degradation. Decimation in the DCT domain avoids aliasing by masking which is a less complicated method than low-pass filtering. The efficiency of the DCT decimation algorithm can be improved by considering only lower frequency DCT coefficients in the blocks to be decimated resulting in small quality degradation in the video. We will explain this later in this section.

For decimation in the DCT domain, an array of DCT blocks first needs to be transformed into one larger DCT block. This process was recently introduced in [13]. However we will show a simpler way to obtain the transformation in a matrix form. Consider we have $N \times N$ DCT blocks of size 8×8 each. To transform them into an $8N \times 8N$ DCT block we need an orthonormal transformation matrix, T^{8N}, i.e. $(T^{8N})^t T^{8N} = I^{8N}$ so that

$$C^{8N} = T^{8N} \begin{bmatrix} C^8_{11} & \cdots & C^8_{1N} \\ & \cdots & \\ C^8_{N1} & \cdots & C^8_{NN} \end{bmatrix} (T^{8N})^t,$$

where C^8_{ij}, for $i, j = 1, \cdots, N$, are 8×8 DCT blocks and C^{8N} is an $8N$ two-dimensional DCT block. Since the transformation matrix T^{8N} is unique for any $\{C^8_{ij}, i, j = 1, \cdots, N\}$ and the corresponding C^{8N}, we can consider the following simple case. Let $C^8_{ii} = I^8$ and $C^8_{ij} = \mathbf{0},\ i \neq j$. From the orthonormal property of the DCT operation matrix given in Eq. (2.8), the expected C^{8N} will be identity matrix I^{8N} as follows

$$I^{8N} = S^{8N} \begin{bmatrix} (S^8)^t S^8 & \cdots & 0 \\ \cdots & (S^8)^t S^8 & \cdots \\ 0 & \cdots & (S^8)^t S^8 \end{bmatrix} (S^{8N})^t,$$

Therefore the transformation matrix is

$$T^{8N} = S^{8N} \begin{bmatrix} (S^8)^t & \cdots & 0 \\ \cdots & (S^8)^t & \cdots \\ 0 & \cdots & (S^8)^t \end{bmatrix} \tag{2.17}$$

with orthonormal property such that $I^{8N} = T^{8N}(T^{8N})^t$. To obtain the forward transformation, let the $8N \times 8N$ DCT operation matrix be separated as follows

$$S^{8N} = \begin{bmatrix} S^{8N}_1 & S^{8N}_2 & \cdots & S^{8N}_N \end{bmatrix} \tag{2.18}$$

Thus, the transformation matrix becomes

$$T^{8N} = \left[T_1^{8N} \quad T_2^{8N} \quad \cdots \quad T_N^{8N}\right], \tag{2.19}$$

and the subblocks $\{T_i^{8N} = S_i^{8N}(S^8)^t\}$ are of size $8N \times 8$. Thus the representation of C^{8N} in terms of the $\{C_{ij}^8\}$, or the forward transformation, is

$$C^{8N} = \sum_{i=1}^{N}\sum_{j=1}^{N} T_i^{8N} C_{ij}^8 (T_j^{8N})^t. \tag{2.20}$$

Hence we have a direct way to obtain an $8N \times 8N$ DCT block from an $N \times N$ array of 8×8 DCT blocks. The inverse transformation of each of the 8×8 DCT blocks is obtained by using the orthonormality of the transformations T_i^{8N} in Eq. (2.20)

$$C_{ij}^8 = (T_i^{8N})^t C^{8N} T_j^{8N}. \tag{2.21}$$

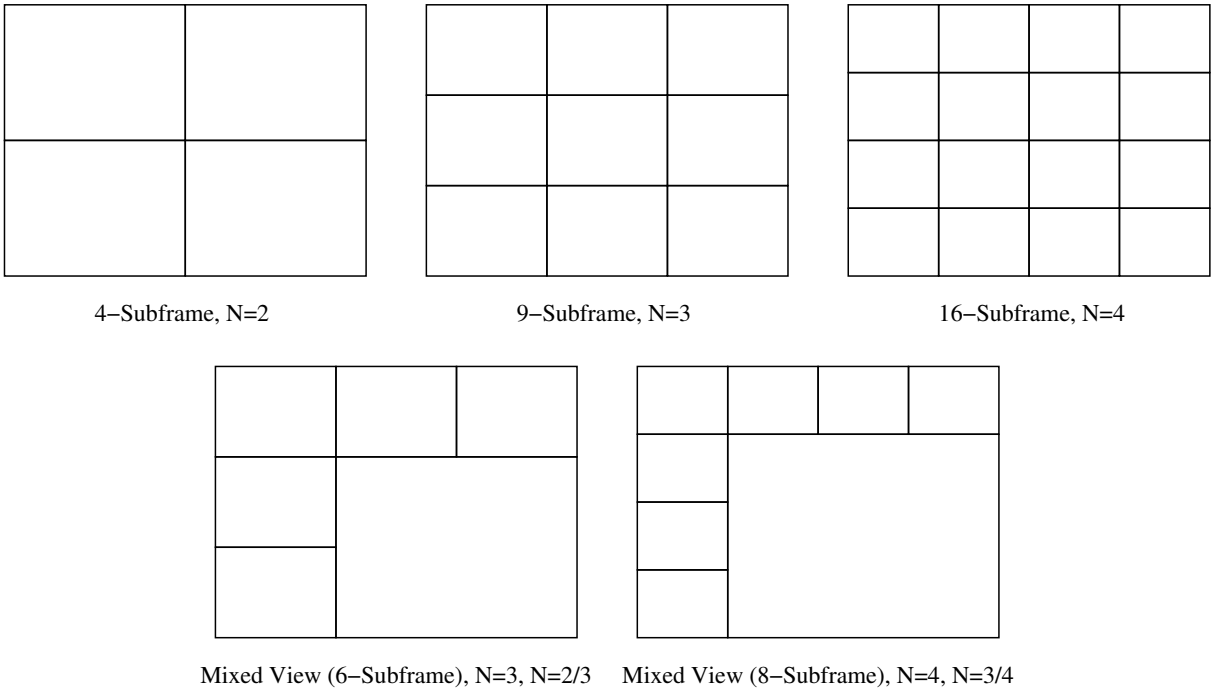


Figure 11: Several video compositing structures

2.2.1 Fast Transformation of DCT Blocks

In this section we will show that the DCT block transformation can be further improved resulting in a faster algorithm. Although the DCT block transformation matrices are sparse, it is possible to derive sparser matrices. As indicated in [12], the odd rows of S_i^{8N}, for $i = 1, .., N$, coincide with the odd rows of S^8, and due to the orthonormality of S^8, the matrices $T_i^{8N} = S_i^{8N}(S^8)^t$, for $i = 1, ..., N$, are such that,

$$JS_1^{8N} = \begin{bmatrix} S^8 \\ Z \end{bmatrix} \tag{2.22}$$

where Z is a "don't care" array, and J is a permutation matrix that separates the odd and the even rows. For simplicity, consider a decimation factor $N = 2$. The results can be extended to other integer factors. From Eq. (2.22) the subblock T_1^{16} is

$$\begin{aligned} T_1^{16} = S_1^{16}(S^8)^t &= J^t \begin{bmatrix} S^8 \\ Z \end{bmatrix} (S^8)^t \\ &= J^t \begin{bmatrix} I^8 \\ Z(S^8)^t \end{bmatrix} \end{aligned} \tag{2.23}$$

displaying that half of its entries are zero or one. It is similar for T_2^{16}, and therefore the transformation matrices are very sparse. Furthermore there is a symmetry between T_1^{16} and T_2^{16} as follows

$$T_2^{16}(i,j) = (-1)^{i+j} T_1^{16}(i,j) \tag{2.24}$$

for $i = 1, \cdots, 16$ and $j = 1, \cdots, 8$. In fact, by definition

$$T_1^{16} = S^{16} \begin{bmatrix} I^8 \\ \mathbf{0} \end{bmatrix} (S^8)^t \tag{2.25}$$

and

$$T_2^{16} = S^{16} \begin{bmatrix} \mathbf{0} \\ I^8 \end{bmatrix} (S^8)^t. \tag{2.26}$$

Also consider $M \times M$ permutation matrices J^M, $M = 8,\ 16$, given by

$$J^M = \begin{bmatrix} 0 & 0 & \cdots & 0 & 1 \\ 0 & 0 & \cdots & 1 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ 1 & 0 & \cdots & 0 & 0 \end{bmatrix}.$$

If we pre- and post-multiply a matrix with the permutation matrices J^{16} and J^8, the matrix is flipped vertically and horizontally, respectively. Therefore

$$\begin{bmatrix} \mathbf{0} \\ I^8 \end{bmatrix} = J^{16} \begin{bmatrix} I^8 \\ \mathbf{0} \end{bmatrix} J^8. \tag{2.27}$$

If we replace Eq. 2.27 into Eq. 2.26, and use the orthonormality properties of DCT matrices $(S^{16})^t S^{16} = I^{16}$ and $(S^8)^t S^8 = I^8$, we will have the following symmetry relationship between T_2^{16} and T_1^{16} as

$$\begin{aligned} T_2^{16} &= S^{16} J^{16} (S^{16})^t S^{16} \begin{bmatrix} I^8 \\ \mathbf{0} \end{bmatrix} (S^8)^t S^8 J^8 (S^8)^t \\ &= K^{16}\ T_1^{16} K^8 \end{aligned}$$

where

$$K^M = \begin{bmatrix} 1 & 0 & \cdots & 0 & 0 \\ 0 & -1 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & 0 & (-1)^{M+1} \end{bmatrix}$$

are the DCT of J^M, $M = 8,\ 16$, as can be easily verified. Due to this symmetry, instead of T_1^{16} and T_2^{16} we consider their sum and difference,

$$\begin{aligned} D_1^{16} &= 0.5\ [T_1^{16} + T_2^{16}] \\ D_2^{16} &= 0.5\ [T_1^{16} - T_2^{16}], \end{aligned} \tag{2.28}$$

which according to the above symmetry gives that $D_1^{16}(i,j) = T_1^{16}(i,j)$ when $(i+j)$ are even and zero otherwise. Similarly, $D_2^{16}(i,j) = T_1^{16}(i,j)$ when $(i+j)$ is odd and zero otherwise.

Hence the sparseness and symmetry of the $\{T_i^{16}\}$ matrices makes the $\{D_i^{16}\}$ matrices very sparse. Therefore forward transformation is very efficient. Replacing the T_i^{16} in terms of the D_i^{16} matrices in the direct transformation we have

$$\begin{aligned} C^{16} &= [X+Y](D_1^{16})^t + [X-Y](D_2^{16})^t \\ X &= D_1^{16}(C_{11}^8 + C_{21}^8) + D_2^{16}(C_{11}^8 - C_{21}^8) \\ Y &= D_1^{16}(C_{12}^8 + C_{22}^8) + D_2^{16}(C_{12}^8 - C_{22}^8) \end{aligned} \tag{2.29}$$

where C^{16} is 16×16 DCT block. Thus we compute the transformation by using the Eq. (2.29), which is computationally less complex than Eq. (2.20) that uses the transformation matrices directly.

Similarly for $N = 3$, 24×24 DCT matrix is obtained such that

$$\begin{aligned} C^{24} &= [X+Y](D_1^{24})^t + [W-Y](D_2^{24})^t + [X-W](D_3^{24})^t \\ X &= D_1^{24}(C_{11}^8 + C_{21}^8) + D_2^{24}(C_{31}^8 - C_{21}^8) + D_3^{24}(C_{11}^8 - C_{31}^8) \\ Y &= D_1^{24}(C_{12}^8 + C_{22}^8) + D_2^{24}(C_{32}^8 - C_{22}^8) + D_3^{24}(C_{12}^8 - C_{32}^8) \\ W &= D_1^{24}(C_{13}^8 + C_{23}^8) + D_2^{24}(C_{33}^8 - C_{23}^8) + D_3^{24}(C_{13}^8 - C_{33}^8) \end{aligned} \tag{2.30}$$

When decimation by a factor $N = 4$, the transformation equations are obtained as the same way as in Equations (2.29) and (2.30).

2.2.2 Improved DCT Decimation

In this section we will first introduce masking of the transformed block, then show an improved decimation algorithm. Again for simplicity consider a decimation factor $N = 2$. It is also possible to extend the results for other integer and rational cases of the decimation factor, N. After obtaining the 16×16 DCT block from 2×2 array of 8×8 DCT blocks, the transformed block is masked to get the top left 8×8 part that includes the low frequency components of the block as follows (see Eq. (2.20)):

$$\begin{aligned} C_d &= [I^8 \;\; \mathbf{0}] C^{16} [I^8 \;\; \mathbf{0}]^t \\ &= \sum_{i=1}^{2} \sum_{j=1}^{2} A_i^8 C_{ij}^8 (A_j^8)^t \end{aligned} \tag{2.31}$$

where $A_i^8 = [I^8 \quad \mathbf{0}]T_i^{16}$, $i = 1, 2$, are 8×8 transformation matrices. If the transformation in Eq. 2.29 is used, the decimated DCT array will be obtained in a less complex manner as

$$\begin{aligned} C_d &= [X_d + Y_d](E_1^8)^t + [X_d - Y_d](E_2^8)^t \\ X_d &= E_1^8(C_{11}^8 + C_{21}^8) + E_2^8(C_{11}^8 - C_{21}^8) \\ Y_d &= E_1^8(C_{12}^8 + C_{22}^8) + E_2^8(C_{12}^8 - C_{22}^8) \end{aligned} \tag{2.32}$$

where 8×8 matrices

$$\begin{aligned} E_1^8 &= [I_8 \quad \mathbf{0}]D_1^{16} \\ E_2^8 &= [I_8 \quad \mathbf{0}]D_2^{16} \end{aligned} \tag{2.33}$$

are sparser than the matrices $\{A_i^8\}$.

To compare the quality of the decimated frame with the original one, we obtain a smooth version of the the decimated frame by interpolation. Consider forward transformation of the 16×16 block

$$\begin{bmatrix} C_d & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} = \sum_{i,j} T_i^{16} \tilde{C}_{ij}^8 (T_j^{16})^t \tag{2.34}$$

where $\{\tilde{C}_{ij}^8\}$ are the smooth-out blocks. Hence applying the inverse transformation in Eq. 2.21, we have

$$\tilde{C}_{ij}^8 = (T_i^{16})^t \begin{bmatrix} C_d & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} T_j^{16} \tag{2.35}$$

for $i, j = 1, 2$.

It is also possible to further improve the decimation by obtaining sparser matrices than the matrices $\{E_i^8\}$. Typically, most of the high frequency coefficients in a DCT block are zero, and even when they are set to zero its inverse DCT values are not very different from

the original ones. Consider then that the DCT blocks to be decimated have $q \times q$ $(1 \leq q \leq 8)$ low-frequency components and the rest are zero

$$C_{ij}^8 = \begin{bmatrix} C_{ij}^q & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \tag{2.36}$$

$$= \begin{bmatrix} I^q \\ \mathbf{0} \end{bmatrix} C_{ij}^q \begin{bmatrix} I^q & \mathbf{0} \end{bmatrix} \tag{2.37}$$

Replacing these blocks in Eq. (2.31) gives

$$C_d^q = \sum_{i,j} B_i^q C_{ij}^q (B_j^q)^t \tag{2.38}$$

where $B_i^q = A_i^8[I^q \quad \mathbf{0}]^t$, $i = 1, 2$, $1 \leq q \leq 8$. Again, these $8 \times q$ matrices are sparse. However, as before if we use these matrices in 2.32, it will become

$$\begin{aligned} C_d^q &= [X_d^q + Y_d^q](F_1^q)^t + [X_d^q - Y_d^q](F_2^q)^t \\ X_d^q &= F_1^q(C_{11}^8 + C_{21}^8) + F_2^q(C_{11}^8 - C_{21}^8) \\ Y_d^q &= F_1^q(C_{12}^8 + C_{22}^8) + F_2^q(C_{12}^8 - C_{22}^8) \end{aligned} \tag{2.39}$$

where

$$\begin{aligned} F_1^q &= E_1^8[I^q \quad \mathbf{0}]^t \\ F_2^q &= E_2^8[I^q \quad \mathbf{0}]^t \end{aligned} \tag{2.40}$$

are sparser than the matrices $\{B_i^q\}$. Also these matrices obviously have fewer entries than the matrices $\{D_i^{16}\}$ and $\{E_i^8\}$. Furthermore, as we decrease q, decimation becomes faster but at the cost of quality. Also the larger q is, the better the interpolation, but the more complex the implementation. When $q = 4$, the complexity of our algorithm, as measured by the number of additions and multiplications, equals to that of [12]. But in the tested images we obtain higher PSNR (Peak Signal to Noise Ratio) values, where PSNR for a 256-gray scale image is given by

$$PSNR = -10log_{10}(\frac{\sum_{i,j}(c_{ij} - \hat{c}_{ij})}{IJ255^2}) \tag{2.41}$$

where c_{ij} and $\hat{c}_{ij}$ are the original and the reconstructed pixel values of an image of size $I \times J$. The improved decimation scheme is illustrated in Fig. 12. Decimation by other factors such as $N = 3$ or $N = 4$ is similar to the decimation by $N = 2$. For example when $N = 3$, a 3×3 array of 8×8 DCT blocks are transformed into one 24×24 DCT block, and then transformed block is masked to obtain the decimated 8×8 DCT block. Like in the case when $N = 2$, it is also possible to represent each 8×8 block with a $q \times q$ part and the rest are zero for faster implementation. The computational complexity depends on q. The quality of the decimated frames is also related to q. For lower values of q, the quality is less but implementation is faster.

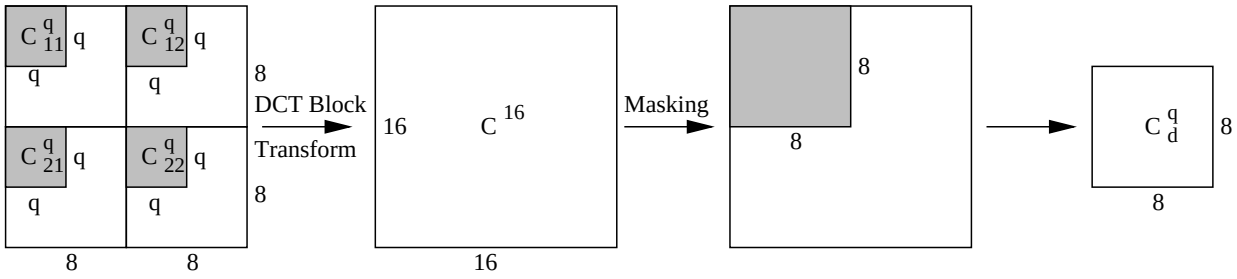


Figure 12: Improved DCT decimation process for $N = 2$

For the rational case, decimation is still possible with higher complexity operations. To obtain the decimated blocks, first an array of DCT blocks are transformed to a larger block which has the size of the sum of the all DCT block sizes. Then masking and re-transformation is applied to the larger block to obtain the decimated 8×8 blocks. For instance, if $N = 2/3$, a 3×3 array of 8×8 DCT blocks are first transformed into a 24×24 blocks. Then the 24×24 DCT block is masked to obtain 16×16 block which is the $2/3$ of it. The additional computational complexity comes from the requirement of obtaining 2×2 array of 8×8 DCT blocks from the masked 16×16 DCT block. The transformation and masking process for the rational case, $N = 2/3$ is shown in Fig. 13.

To see the performance of our algorithms we compared the computational complexity of ours and that of other algorithms. We also obtained some PSNR values in dB to compare the

video quality of our and the other methods. In Table 2, we illustrate the number of additions and multiplications per pixel for our algorithm and other three algorithms: spatial, of Chang et al. [5] and of Dugad et al. [12] for $N = 2$. As seen from the table, the computational complexity of our algorithm is equal to that of the Dugad et al.'s [12] which requires the least computation.

For comparison of the video quality of our algorithm and Dugad et al.'s we obtain PSNR values for some video frames. In Table 3, we show the results for six frames from different video sequences. At the same computational complexity, which is the case when $q = 4$, our algorithm always gives slightly better results than those of Dugad et al.'s [12] for all video frames except the frame from Claire sequence. As shown in the table, our algorithm with $q = 8$ always performs better with a slight increase in number of computations. However, the case of $q = 4$ is enough to obtain better results without an increase of computational complexity.

The computational complexity of our algorithms for the integer decimation factors and also for the rational case of 2/3 are provided in Table 4. In this table, direct method is the one that uses the matrices $\{T_i^{8N}\}$ in Eq. (2.20). For a given decimation factor N, the improved decimation method with q=8, which requires less computations, gives exactly the same results with the direct method in terms of PSNR. In Fig. 14, we show some results for decimation factors N =2, 3, 4, and 2/3 respectively. As seen from these figures, even if q is less then 8 we still have good PSNR values compared to the case of $q = 8$. For the decimation factor $N = 4$, the results of the two cases of $q = 8$ and $q = 4$ are so close that the lines of PSNR values coincide in the figure. It is also possible to use smaller q to decrease the number of additions and multiplications. However this will affect the quality to get worse.

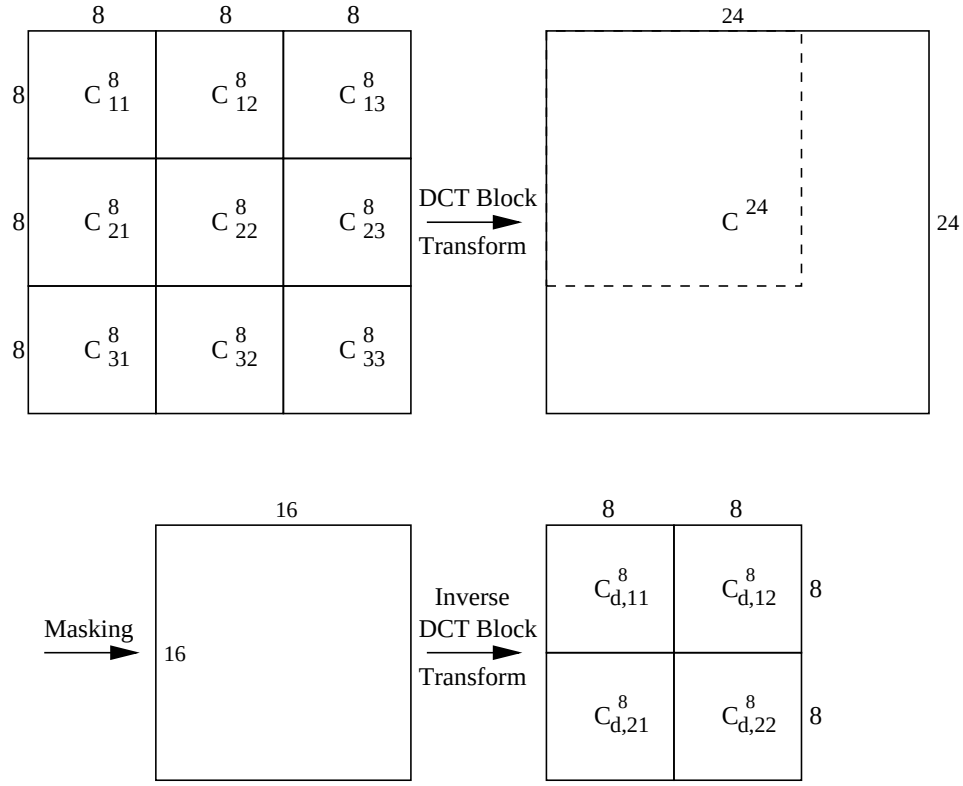


Figure 13: DCT decimation for the rational case, $N = 2/3$

Table 2: Computational complexity comparisons of four decimation methods for $N = 2$

Method	Multiplications/Pixel	Additions/Pixel
Spatial	3.44	9.82
Chang's	4.00	4.75
Dugad's	1.25	1.25
Ours (q=4)	1.25	1.25

Table 3: PSNR comparisons of the decimation methods for $N = 2$

Video Frame	Dugad's	Ours (q=4)	Ours (q=8)
Miss America	39.1569	39.2989	39.6038
Salesman	30.6821	30.7773	31.1708
Foreman	32.6638	32.7484	33.1292
Hall	28.5138	28.6535	29.0496
News	29.7323	29.8972	30.3730
Claire	33.6414	33.5041	33.7074

Table 4: Computational complexity of our decimation algorithms

Decimation Factor (N)	Method	Multiplications/Pixel	Additions/Pixel
2	q=4	1.25	1.25
	q=8	3.38	4.13
	Direct	9.00	7.75
3	q=3	0.91	0.94
	q=8	3.83	4.72
	Direct	9.08	7.97
4	q=2	0.44	0.44
	q=4	1.22	1.41
	q=8	3.91	4.84
	Direct	12.50	11.44
2/3	q=6	18.38	17.16
	q=8	21.58	20.81
	Direct	39.25	35.47

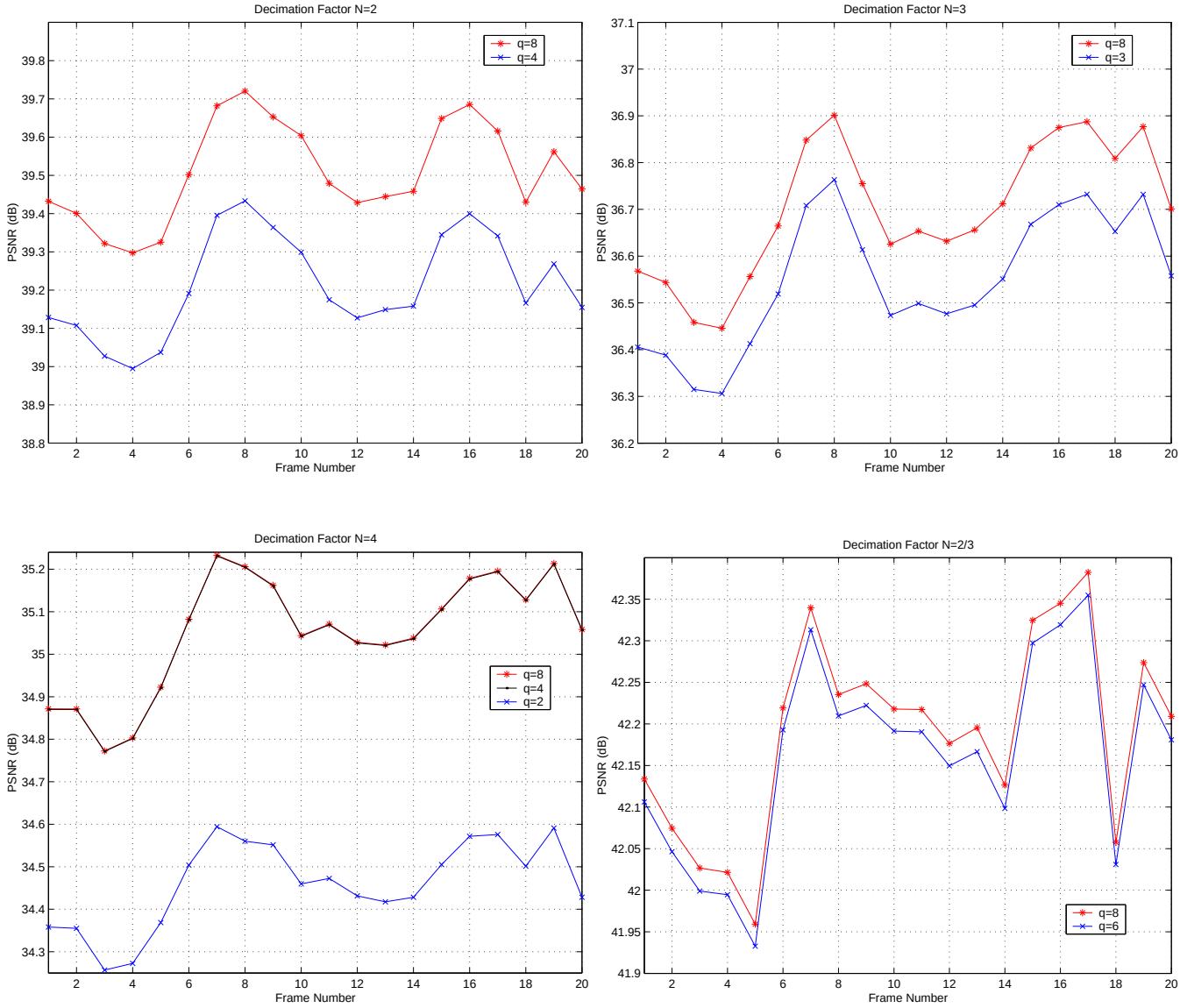


Figure 14: PSNR comparisons of different decimation factors for Miss America sequence

3.0 EMBEDDED ZEROTREE CODING OF DCT COEFFICIENTS

In this chapter we will investigate DCT-based embedded zerotree coding. Zerotree coding is a progressive coding method which encodes a video or image into a bit stream with increasing precision. The embedded property is accomplished that all encodings of the same image or video at lower bit rates are embedded in the beginning of the bit stream for the target bit rate [15]. The embedded coding scheme depends on coding a symbol by an entropy encoder as soon as the symbol is obtained by the zerotree coding. A zerotree is a tree whose leaves correspond to the insignificant transform coefficients which are less than a certain threshold. A zerotree can be encoded by a single symbol resulting in efficient coding. Zerotree coding proceeds iteratively producing at each iteration a significance map of all coefficients. Thus more generally zerotree coding is called significance tree quantization.

3.1 INTRODUCTION

As video applications continue to grow, significance tree based image compression techniques are becoming more effective and less complex. One of these methods, embedded image coding using zerotrees of wavelets, was first introduced by Shapiro in 1993 [15]. Dependencies of wavelet coefficients in subbands are well exploited in this method. Later, beside wavelets, DCT-based zerotree coding applications were developed and used by several researchers [18, 19, 20, 33]. This more recent work shows that DCT-based embedded coders can provide competitive compression rates with a good image quality compared to the wavelet based embedded coders. As a progressive coding method, an embedded zerotree encodes the largest, most important coefficients first. In this manner, the decoder first receives the coefficients

that have the largest content of information yielding the largest distortion reduction. Embedded bitstream obtained by adaptive arithmetic coder representing the symbols of the zerotree coding indicates the ordered coefficients by magnitude. To measure the distortion between the original and reconstructed transform coefficients we consider Mean Square Error given by

$$MSE_C(C - \hat{C}) = \frac{1}{IJ} \sum_i \sum_j (C_{ij} - \hat{C}_{ij})^2 \tag{3.1}$$

where C_{ij} and $\hat{C}_{ij}$ are the original and the reconstructed transform coefficients of an image or video frame of size $I \times J$ respectively. In a progressive transmission scheme, the decoder initially sets the reconstruction coefficients $\{\hat{C_{ij}}\}$ to zero and updates them according to the incoming symbols. After receiving the approximate or exact values of some transform coefficients the decoder can reconstruct the video frame. From Eq. (3.1), it is clear that if the exact or approximate value of the transform coefficient C_{ij} is sent to the decoder, the MSE_C of the reconstructed frame decreases. This means that the larger transform coefficients should be sent first because of their larger content of information.

Beside the progressive property there is another advantage of the embedded zerotree coding. Since the embedded zerotree encoder can be stopped at any time, it is very easy to reach the exact target bit rate or the desired quality of image or video without truncating the lower part of a video frame as in other methods. Analogously, a decoder can cease at any point where the desired quality or the bit rate is reached.

In the following sections, the idea of the DCT-based embedded zerotree coding and the details of this method are given and illustrated. The adaptive arithmetic coding is given in Appendix B.

3.2 DEFINITION AND FEATURES OF DCT-BASED EMBEDDED ZEROTREE CODING

The embedded zerotree coding of DCT coefficients is based on four steps: (1) arranging DCT coefficients into hierarchical scales similar to the wavelet subband structure, (2) determining the significant coefficients across scales by exploiting the self-similarity inherent in DCT coefficients, (3) successive-approximate quantizing of DCT coefficients, (4) lossless compressing of the data from the output of the embedded zerotree coder by using an adaptive arithmetic coder.

C_1	C_2	C_5	C_6	C_{17}	C_{18}	C_{19}	C_{20}
C_3	C_4	C_7	C_8	C_{21}	C_{22}	C_{23}	C_{24}
C_9	C_{10}	C_{13}	C_{14}	C_{25}	C_{26}	C_{27}	C_{28}
C_{11}	C_{12}	C_{15}	C_{16}	C_{29}	C_{30}	C_{31}	C_{32}
C_{33}	C_{34}	C_{35}	C_{36}	C_{49}	C_{50}	C_{51}	C_{52}
C_{37}	C_{38}	C_{39}	C_{40}	C_{53}	C_{54}	C_{55}	C_{56}
C_{41}	C_{42}	C_{43}	C_{44}	C_{57}	C_{58}	C_{59}	C_{60}
C_{45}	C_{46}	C_{47}	C_{48}	C_{61}	C_{62}	C_{63}	C_{64}

Figure 15: Treating an 8×8 DCT block as a 3-scale subband structure

Consider a video frame which is composed of $K \times L$ blocks with sizes of $M \times M$, where each block is 2-D DCT transformed. Each DCT block of size $M \times M$, including M^2 coefficients, can be treated as a hierarchical subband structure. In Fig. 15, we show an 8×8 DCT block with its coefficients treated as 3-scale subband structure and ordered according to raster scanning, which we will see later. Rearranging all blocks of the frame in this way, a 3-scale hierarchical subband structure of a DCT frame, which can be seen in Fig. 16, is obtained. In

Fig. 16, the subband LL_3 includes the DC coefficients of all 8×8 DCT blocks. It is identical with the highest subband of a wavelet structure. In this layer, the number of coefficients is equal to the number of DCT blocks of the video frame. All other subbands include AC coefficients. Since most of the energy is concentrated in the DC coefficients, the quality of the decoded image depends mostly upon DC coefficients, then on the AC coefficients. Therefore the rearranged DCT structure is suitable for the zerotree encoding, and flexible to control the bit rate [33].

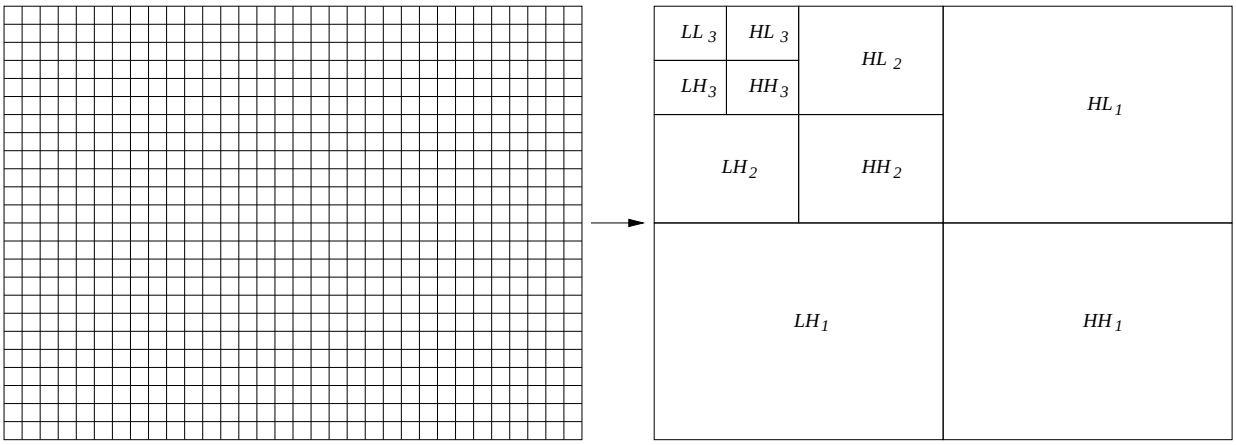


Figure 16: Conversion of an 8×8 DCT-frame into 3-scale subband frame

Several rearrangements of DCT blocks other than 3-scale structure are also possible [19, 20]. Individually each one gives comparably good results in terms of the compression ratio and quality [33]. In Fig. 17, we display an example of an $M \times M$ DCT-blocks frame and its rearranged version of 3-scale structure.

In the hierarchical subband structure of a DCT frame, from a coarser to the next finer scale, a relationship can be established between the coefficients of similar orientation forming a tree structure. If a coefficient at a given coarse scale is called 'parent', all the coefficients at the next finer scale in the same spatial location of similar orientation are called 'children'. Specifically, for a given child at a fine scale, all coefficients at the coarser scales of similar

orientation at the same spacial locations are called 'ancestors'. Similarly, for a given parent at a coarse scale, all coefficients at finer scales of similar orientation are called 'descendants'. This parent-child relationship is shown in Fig. 18. In this example, while each coefficient at LL_3 subband has three children, coefficients of LH_3, HL_3, HH_3, LH_2, HL_2, and HH_2 subbands have four children each. During the zerotree coding, each parent is scanned before its children. In Fig. 18, the dotted lines show the scanning order of the subbands, and each small square block represents a DCT coefficient.

Zerotree coding depends on transmitting the positions of significant and insignificant coefficients. After arranging DCT coefficients into 3-scale subband structure, a significance test is performed. Zerotree maps indicating the positions of the significant and insignificant coefficients are called significance maps. Zerotree coding ensures a compact multi-resolution representation of significance maps [15].

A DCT coefficient C is said to be significant with respect to a given threshold Th, if its magnitude is bigger than the given threshold, i.e., $|C| > Th$. There are four symbols used in zerotree coding: (1) T: zerotree root , (2) Z: isolated zero, (3) P: positive significant coefficient, (4) N: negative significant coefficient. If a parent and all its descendants are insignificant with respect to a given threshold, then the parent is called a zerotree root. Instead of coding all elements of a zerotree, only the zerotree root is encoded representing that the insignificance of the other elements at finer scales are entirely predictable. If a cofficient at a coarser scale is insignificant, and at least one of its descendants is significant, the coefficient at the coarser scale is encoded as an isolated zero. If a coefficient is significant, it is encoded either as positive or negative according to its sign.

Beside the scanning order of the subbands, the obtained zerotree symbols are scanned according to a predetermined scan path at each subband. With this information, the decoder will be able to reconstruct the encoded signal by using the same scanning path. Three scanning examples of the zerotrees using raster, Morton, and Peano methods [23] are shown in Fig. 19, respectively.

In zerotree coding, coefficients are ordered due to their significance by using successive approximation quantization, which is explained in the next section.

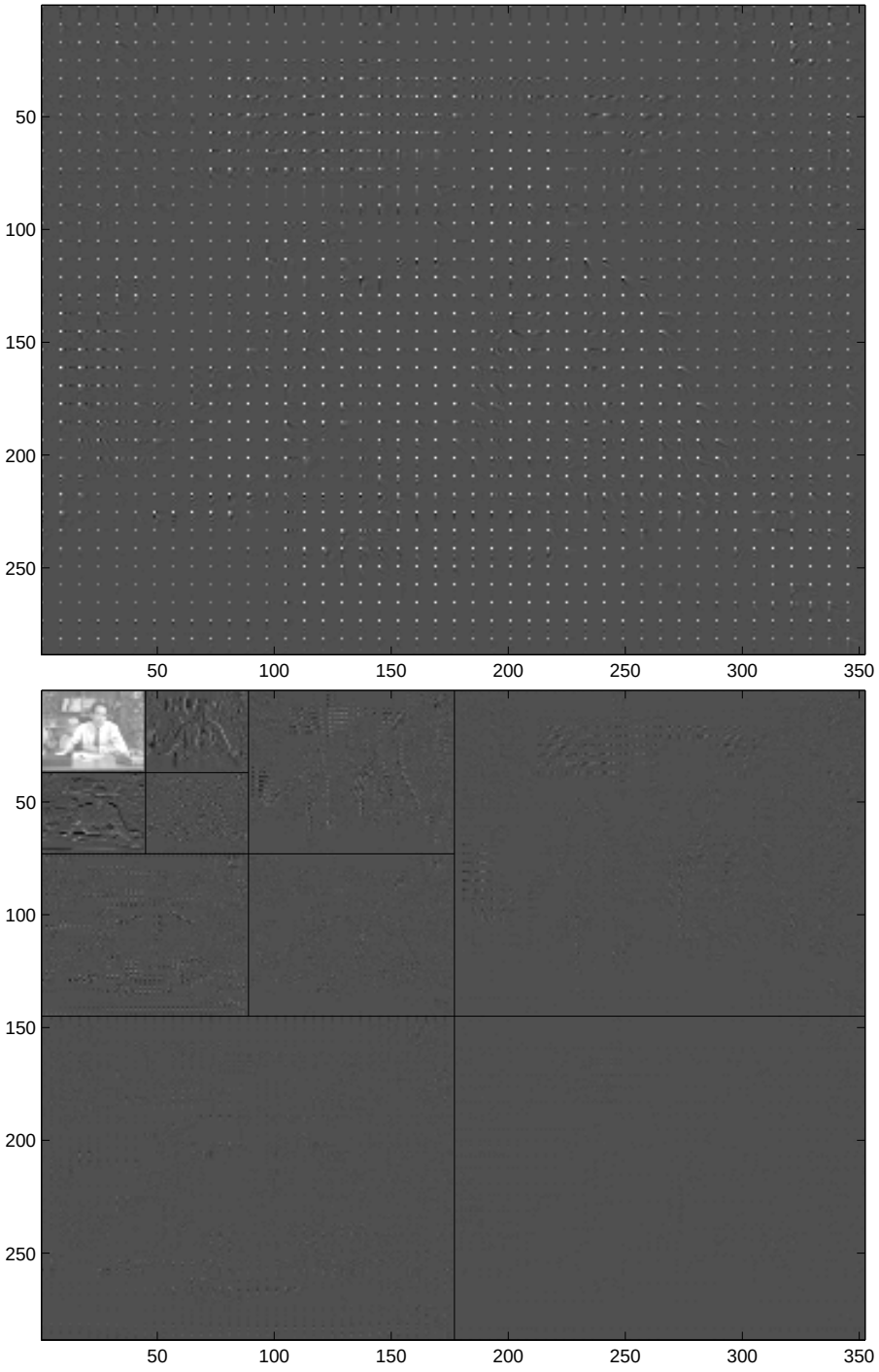


Figure 17: An 8×8 DCT-blocks frame, and its rearranged version of 3-scale subband structure

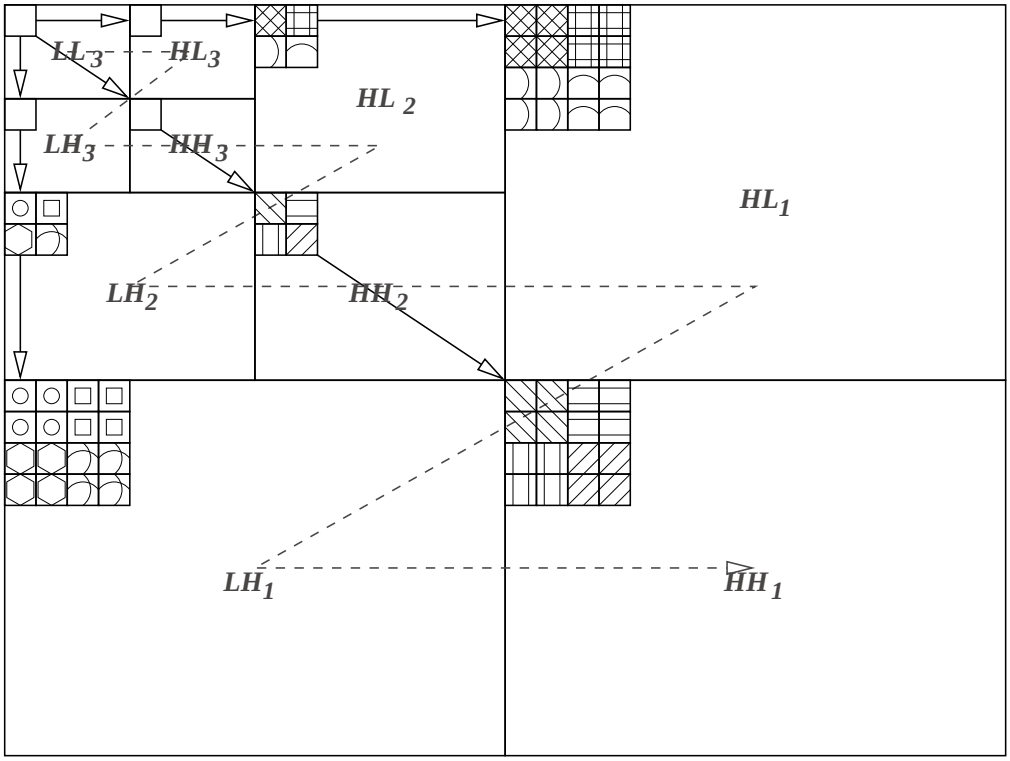


Figure 18: Parent-child relationship of 3-scale DCT subband structure

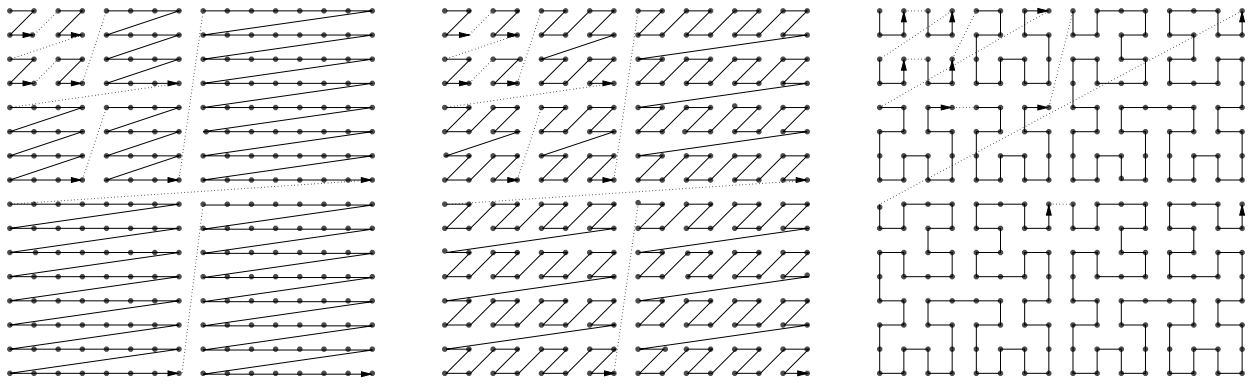


Figure 19: Raster, Morton, and Peano scan paths of a 3-scale subband structure

3.3 SUCCESSIVE APPROXIMATION QUANTIZATION

Successive approximation quantization (SAQ) is implemented in two consecutive passes. At each pass, it produces embedded code parallel to the binary representation of an approximation to a real number [15]. The SAQ is applied iteratively for each new threshold. The initial threshold Th_0 is chosen as

$$Th_0 = 2^n \tag{3.2}$$

where $n = \lfloor log_2 C_{max} \rfloor$, and $C_{max} = max(C_{i,j})$ for $i = 1, ..., I$ and $j = 1, ..., J$, for an $I \times J$ DCT frame. Starting from Th_0, at each successive step the other thresholds are obtained according to $Th_i = Th_{i-1}/2$, $i \geq 1$. For each threshold two passes are performed: dominant pass and subordinate pass. They will be detailed in the following two subsections.

3.3.1 Dominant Pass

Dominant pass is an implementation of the zerotree coding. The dominant pass is performed from the coarsest to the finest subband (see the dotted lines in Fig. 18). During the dominant pass, the set of coordinates of insignificant coefficients, which is called dominant list, is used. Initially, all DCT coefficients are considered as insignificant and put in the dominant list. Coefficients with coordinates on dominant list are compared with the threshold Th_i. If a coefficient is found to be significant, its sign is determined. The obtained significance map is zerotree coded as explained in the previous section. The magnitudes of the coefficients which have been found to be significant during the dominant pass are removed from the dominant list and put in subordinate list, which is the topic of the next subsection. To avoid the occurrence of these coefficients on future dominant passes, they are replaced with zeros in the DCT frame. Significant coefficients determined during a dominant pass are reconstructed in the decoder according to $\hat{C} = 1.5 \times Th_i$, corresponding the center of the uncertainty interval $[Th_i, Th_{i-1})$.

3.3.2 Subordinate Pass

The magnitudes of the coefficients found to be significant are now the contents of the significant list. After the dominant pass, to add more precision to the quantized DCT coefficients, a subordinate pass is performed. For the subordinate pass, the width of the quantization step size is cut in half. More clearly, cutting in half a previous uncertainty interval, $[In_a, In_b)$, two new uncertainty intervals, $[In_a, In_m)$ and $[In_m, In_b)$, where $In_m = median(In_a, In_b)$, are obtained. All of the previous intervals are halved in this way. Subordinate pass refines the significant coefficients by setting them as the center of one of the new intervals, adding a precision of one bit. If a significant coefficient is in the lower interval a "0" symbol, if it is in the upper one "1" symbol is generated for the refinement.

By using the dominant pass, the coefficients are automatically ordered in importance. However, since the coefficients on the subordinate list are sent to the decoder in the same scan order of the dominant list, they are not ordered according to their magnitude. In this case while adding negligible complexity, it increases coding efficiency.

The passes alternate between dominant and subordinate passes until either the desired bit budget or quality is reached. Stopping the encoding of an embedded bit stream at any point gives a precise rate control. However this is not the case for non-embedded coders, which results in a truncation at the bottom part of the video frame. The flowchart of the SAQ algorithm is shown in Fig. 20.

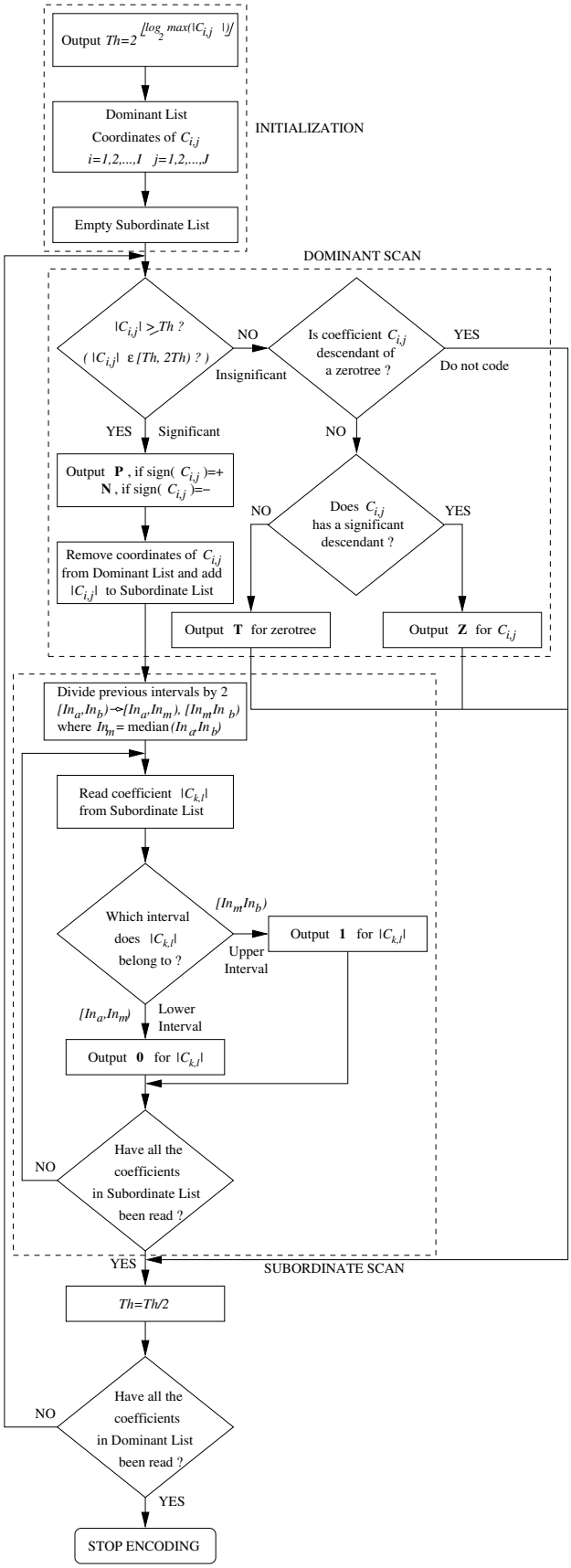


Figure 20: Flowchart of zerotree coding

3.4 AN EXAMPLE

Consider the 8×8 DCT block of an image shown on the top left of Fig. 21. The other blocks on the first column of the figure are modified versions of the DCT block where significant coefficients are replaced with zeros during the previous dominant passes. In this example, four dominant and subordinate passes of embedded zerotree coding are shown. To give more precision to the coefficients, the number of passes can be increased. The initial threshold is founded to be 64 according to Eq. (3.2). According to the first threshold, only two coefficients, 109 and -75, are significant. The symbols for these significant coefficients are P, and N, respectively. Since the parent 21 at the scale HL_3, its children at the scale HL_2, and its grandchildren at the scale HL_1 are all less than 64, a zerotree T symbol stating the insignificance of the HL family is generated and is put at the coordinates of the ancestor 21. Another T symbol is generated for the LH family, since all of them are insignificant. The HH family has a different structure because of the significant parent -75, which is encoded with the symbol N. The parents at the HH_2 have T symbols individually since their children are insignificant like they are.

The significance map obtained and shown in the second column is scanned by using one of the methods shown in Fig. 19. The raster scan is used. Then zerotree array of first dominant pass will be $PTTNTTTT$. This array is encoded by using a four-symbol adaptive arithmetic encoder. Essentially, as soon as a symbol is generated, it is encoded by an adaptive arithmetic encoder for embedded bit propose and due to the possibility of ceasing encoding at any point. Since the first uncertainty interval is $[T_0, 2 \times T_0) = [64, 128)$ and the center of it is 96, the significant coefficients of the first dominant pass are decoded as 96, and -96 at the decoder. The blocks at the third column display the reconstruction of the DCT block after each dominant pass. The positions of the significant coefficients 109, and -75 are removed from the dominant list. For the next possible passes they are replaced with zeros so as not to be compared with the smaller thresholds.

After the dominant pass is performed, two significant coefficients which their magnitudes are appended to subordinate list are compared with a decision boundary. Subordinate pass cuts in half the uncertainty interval into two new uncertainty intervals, $[64, 96)$ and $[96, 128)$.

Thus decision boundary here is 96. Since 109 is bigger than the decision boundary 96, it is quantized to 112 which is the center of the upper interval, and will be encoded with the symbol "1". The magnitude of -75 is in the lower interval. Thus it is encoded with the symbol "0", and its quantized value is refined from 96 to the center of first interval, 80. The refined DCT block after the first subordinate pass is shown at the last column of the first line of the figure. The scanning order of subordinate list is the same as the dominant list. Thus the array of the first subordinate pass is "10". This is encoded by a 2-symbol adaptive arithmetic coder.

At the second row of the figure, the first block has zeros in the coordinates of significant coefficients from the first dominant pass. The coefficients of the modified DCT block is now compared with the new threshold 32. In this case, 33, 61, and -43 are significant coefficients. Thus, the symbols P, P, and N are generated respectively. Since the ancestor of the all coefficients, DC coefficient, is now zero and three of its descendants are significant, Z symbol is generated for the DC coefficient. The other symbols for the other coefficients are shown in the second column of the second row. Since we add a new interval at each dominant pass, the significant coefficients are decoded to 48 at the decoder, which is the center of the new interval $[32, 64)$. After raster scanning of the zerotree symbols, array $ZZTZPTTTZTTTTPTTNTT$ is formed.

Three significant coefficients are appended to the subordinate list, giving total five coefficients to be refined by the subordinate pass. Until now, there are three intervals obtained. The second subordinate pass divides each interval into two new intervals. Thus there exists total of six intervals for the subordinate pass, which are $[32, 48)$, $[48, 64)$, $[64, 80)$, $[80, 96)$, $[96, 112)$, and $[112, 128)$. For this example, all of the intervals, centers, and the symbols for subordinate passes are shown in Fig. 22. The arrowheads show the centers of the intervals. The second subordinate pass obtains "00001" array at the output. The third and fourth dominant and subordinate pass results and zerotrees are also shown in Fig. 21. Additional passes are also possible in order to achieve better quality.

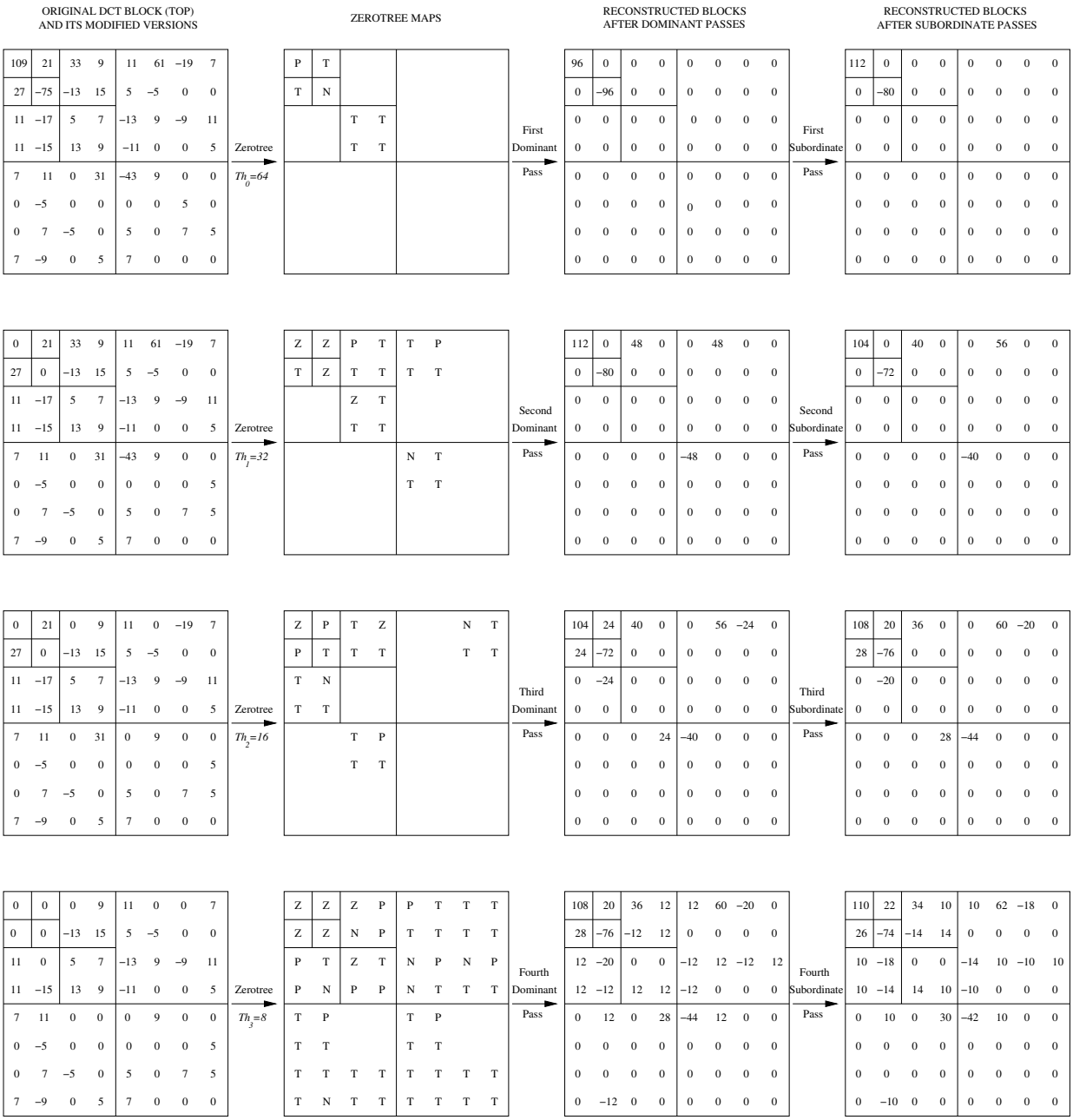


Figure 21: An example of zerotree coding

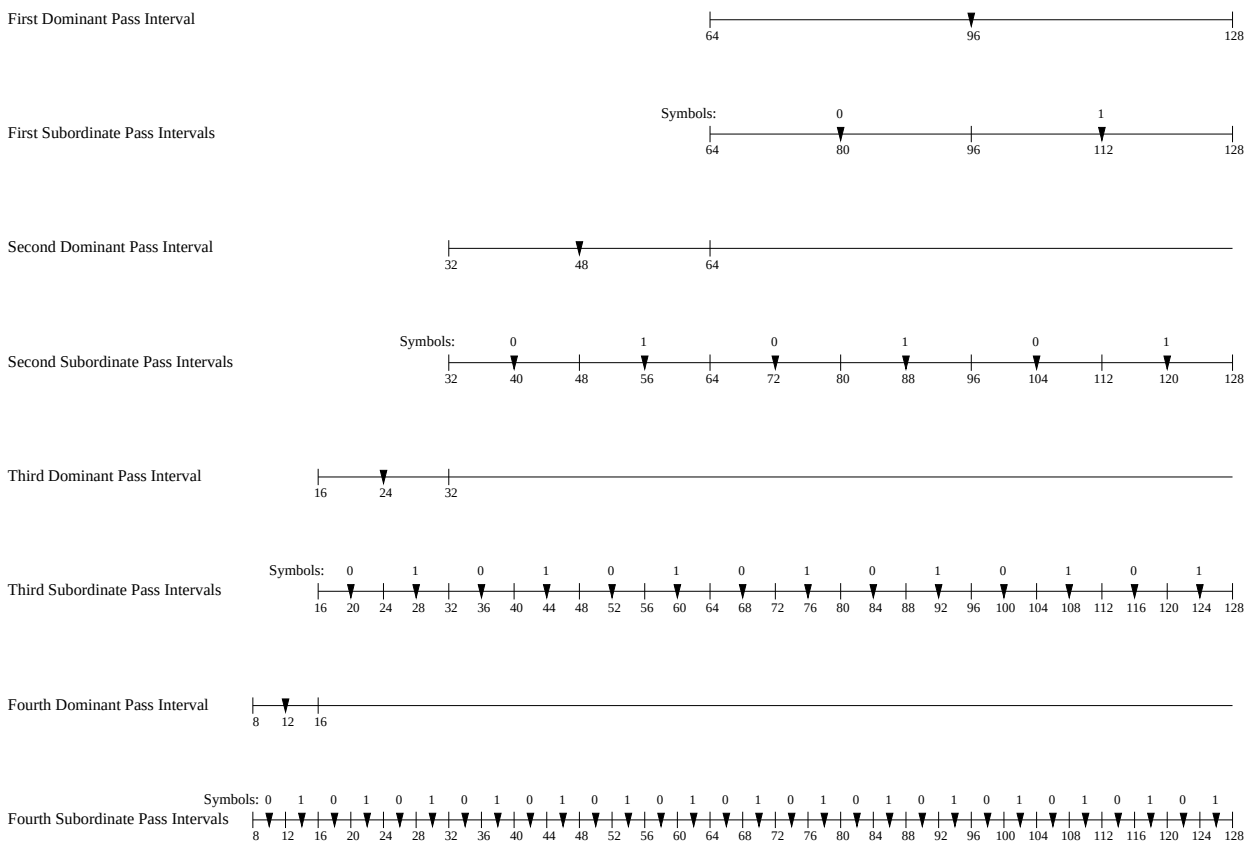


Figure 22: Dominant and subordinate pass intervals

3.5 EXPERIMENTAL RESULTS

In the experiments, we first use the conventional DCT domain video compositing system, which exploits regular scalar quantization, in Fig. 4 with several decimation factors. Then we replace it with the DCT-based embedded zerotree (DCT-EZT) coder. We compare the results of both compositing systems in terms of PSNR at the same bit rate to see the quality improvement by the DCT-EZT coding. For the conventional DCT compositing, the adaptive arithmetic coder is used instead of the conventional Huffman coder to permit a fair comparison. The first video frame is coded as intra (I), and rest are coded as interframes (P) forming a typical structure of group of pictures (GOP).

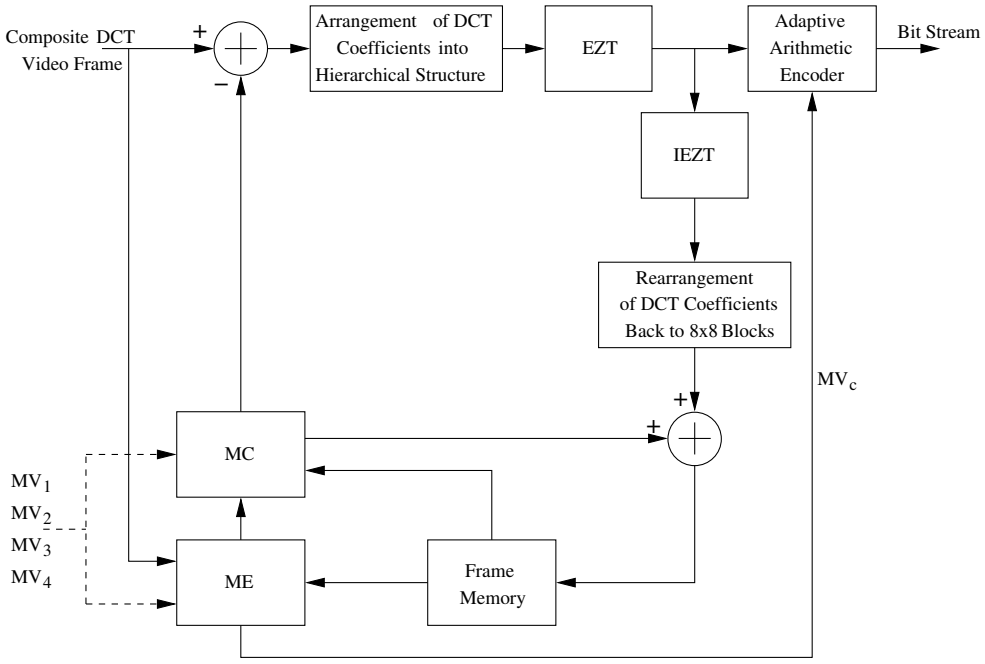


Figure 23: DCT-EZT encoder

To use the DCT-EZT encoder with the proposed compositing, DCT coefficients are rearranged into hierarchical structure with ten subbands as explained in Chapter 3. We also subtract the average of the DC values in the coarsest subband, LL_3, and transmit as

an overhead in order to improve the coding efficiency. Because the correlation of the DC coefficients of neighboring blocks is very high this method decreases the unnecessary scanning of the zerotrees [20]. We also treat the coefficients in the coarsest subband as if they do not have any children. Accordingly, after zerotree coding of the coarsest subband we obtain an array consisting of three symbols, Z, P, and N for this subband, in this way we do not use the four symbols-alphabet for all zerotree coding. This improves the coding efficiency of adaptive arithmetic coder. Furthermore, all other symbols in the finer subbands except the ones in the finest scales are encoded by using the four symbol-alphabet adaptive arithmetic coder. In the finest scale subbands, there are only three symbols since the coefficients do not have any children. Thus the zerotree array of these subbands are encoded by a three symbols-adaptive arithmetic encoder.

Then the binary symbols obtained from the subordinate pass are encoded by two-symbol alphabet adaptive arithmetic encoder. The dominant and subordinate passes are subsequently used until the desired bit budget is reached. For motion estimation and compensation, DCT coefficients are inverse EZT coded and reformed to their initial 8×8 block structure. Motion compensation is done by either estimating the motion vectors from the incoming streams, or by computing the motion vectors directly from the composite video. The initial threshold, Th_0, is also sent to the decoder. Then the decoder starts to decode incoming zerotree symbols according to Th_0 as the way explained in the example in Section 3.4. The proposed DCT-EZT encoder is shown in Fig. 23.

Before comparing the conventional and DCT-EZT compositing systems we first investigate if the scanning path of the EZT coefficients have any influence on the final compression result or quality. For this, we compare three of the scan paths, raster, Morton and Peano, which are shown in Fig. 19. In this comparison, we use four different video sequences, Claire, Miss America, Salesman, and Trevor, in CIF format, which have the size of 288×352, and decimate them with the decimation factor, $N = 2$. After compositing the four decimated frames into one, we encode the composited video frames by using DCT-EZT encoder with using each of three scanning methods. In Table 5, we illustrate the average PSNR results of seventy frames for each scanning methods. We also show the individual PSNR values in Fig. 24. As seen from Table 5 and Fig. 24 there is no major effect of the scanning methods

Table 5: Average PSNR values obtained from three scan methods according to given constant bit rates

Bit Rate (bits/frame)	Raster Scan	Morton Scan	Peano Scan
20000	34.9523 dB	34.9524 dB	34.9480 dB
40000	38.5217 dB	38.5275 dB	38.5170 dB
60000	39.2390 dB	39.2288 dB	39.2525 dB
80000	40.6408 dB	40.6315 dB	40.6327 dB

on the performance. Thus we chose raster scan to use with the proposed DCT-EZT coder for the rest of the experiments.

For the comparison of conventional DCT compositing and DCT-EZT based compositing methods, we use two different decimation methods, one using integer factor and the other a rational factor. For the integer case, $N = 2$, four video sequences are composited into one, while in the mixed-view six video sequences are composited into one, five videos being decimated by 3, and the other decimated by a rational number, 2/3. In both cases, we obtain better results by using the DCT-EZT coding than by using the conventional DCT encoder. For conventional DCT encoder we use four different Quantization Parameters (QP), which are QP =3, 5, 8, and 10. We also use syntax based adaptive arithmetic coder to encode the symbols obtained from the conventional DCT encoder. These symbols in the block layer are the combination of (LAST, RUN, LEVEL), namely TCOEFs (Transform Coefficients), and individually, LAST, RUN, LEVEL, SIGN for both intra- and interframe, and INTRADC for intraframe. The LAST symbol is the indication of the remaining non-zero coefficients in a DCT block. If LAST is 0, there are more non-zero coefficient(s) in the DCT block, if 1, it means that this is the last non-zero coefficient in the block, thus there is no need to look further for the other coefficients in the block. The RUN symbol stands for the number of successive zeros preceding the coded coefficient. The LEVEL is the non-zero value of the quantized coefficient. These symbols and the initial cumulative frequencies for adaptive arithmetic coding were taken from H.263 recommendation in [26]. For the

most commonly occurring events a table of combinations of (LAST, RUN, LEVEL), which are called TCOEFs, were used. Unlike Huffman coding, the predetermined variable length codes (VLC) of each TCOEF with fixed length, which are supplied in [26], are not used in the adaptive arithmetic coding case. Consequently adaptive arithmetic coding results in better performance. For the remaining combinations of LAST, RUN, and LEVEL, they are coded separately. Thus for each of them different histograms are used to track the changing probabilities of the symbols. The signs of the coefficients, SIGN, and the itraframe DC coefficients, INTRADC are also encoded in the same way by using their own histograms.

In the Fig. 25, we show the PSNR results of compositing four video streams (N=2) each with seventy frames for both DCT-EZT and conventional DCT encoder using the changing bit rate given by the regular quantizer with quantization factors, QP=3, 5, 8, and 10. Some of the composite video frames from each conventional DCT and DCT-EZT encoder systems are shown in Fig. 26. We also show the PSNR results for the mixed-vied composited videos (N=3, 2/3) in Fig. 27. The mixed-view video samples are shown in Fig. 28.

As seen from both PSNR comparisons and subjective tests of video frames the proposed DCT compositing with DCT-EZT encoder outperforms the DCT compositing system with the conventional DCT encoder using regular scalar quantizer. The average PSNR values for the four composited and mixed view video sequences are shown in Tables 6 and 7 respectively.

Table 6: Average PSNR values for four composited videos ($N = 2$)

	QP=3	QP=5	QP=8	QP=10
Conventional DCT Encoder	39.0713	36.5549	34.3038	32.8692
DCT-EZT Encoder	40.9160	38.7932	36.8879	35.4929

Table 7: Average PSNR values for mixed-view composited videos ($N = 3$, 2/3)

	QP=3	QP=5	QP=8	QP=10
Conventional DCT Encoder	38.5867	35.5643	33.0410	31.5754
DCT-EZT Encoder	40.4920	37.9544	35.1252	34.0327

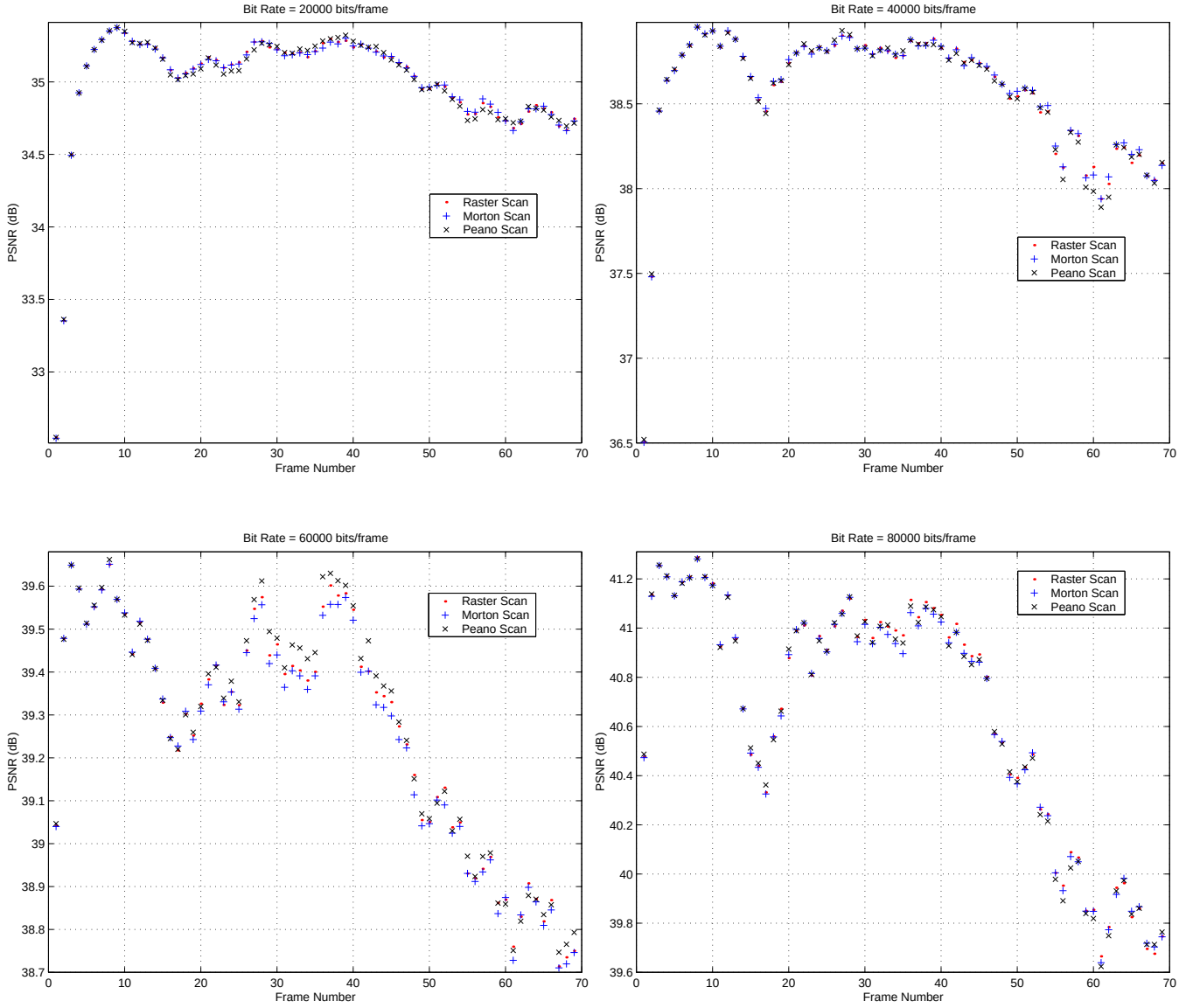


Figure 24: PSNR comparisons of DCT-EZT coder for Raster, Morton and Peano scan methods

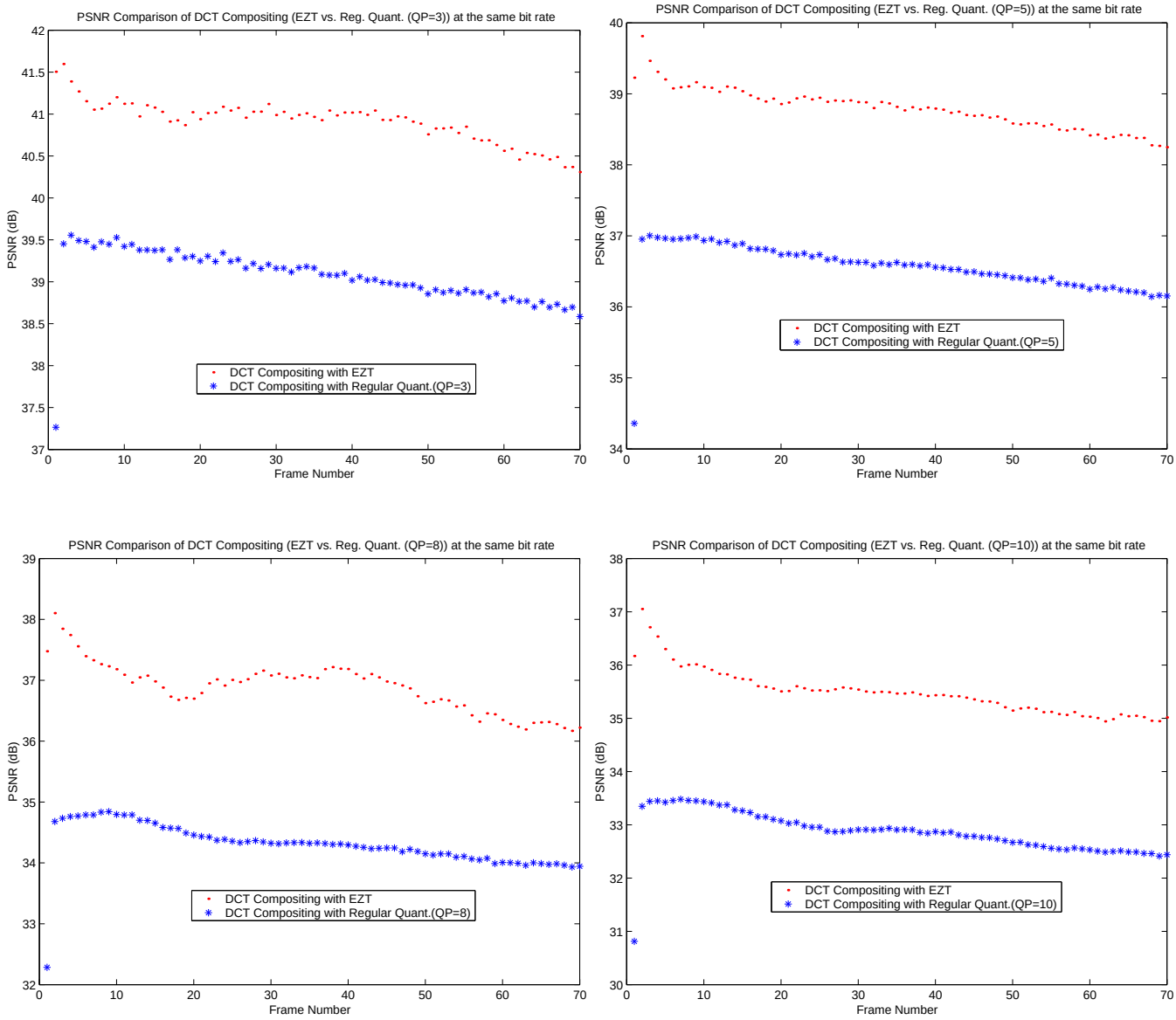


Figure 25: PSNR comparisons of DCT-EZT and conventional DCT encoder for four composited videos

(a) Original composited frame (Frame # 5)

(b) Conventional DCT (QP=3), 74352 bits
PSNR=39.4793 dB

(c) DCT-EZT, 74352 bits
PSNR=41.1532 dB

Figure 26: Composited video frame samples from the conventional DCT and the DCT-EZT codings

(d) Conventional DCT (QP=5), 51480 bits
PSNR=36.9644 dB

(e) DCT-EZT, 51480 bits
PSNR=39.2019 dB

(f) Conventional DCT (QP=10), 30042 bits
PSNR=33.4238 dB

(g) DCT-EZT, 30042 bits
PSNR=36.3004 dB

Figure 26 (Cont.): Composited video frame samples from the conventional DCT and the DCT-EZT codings

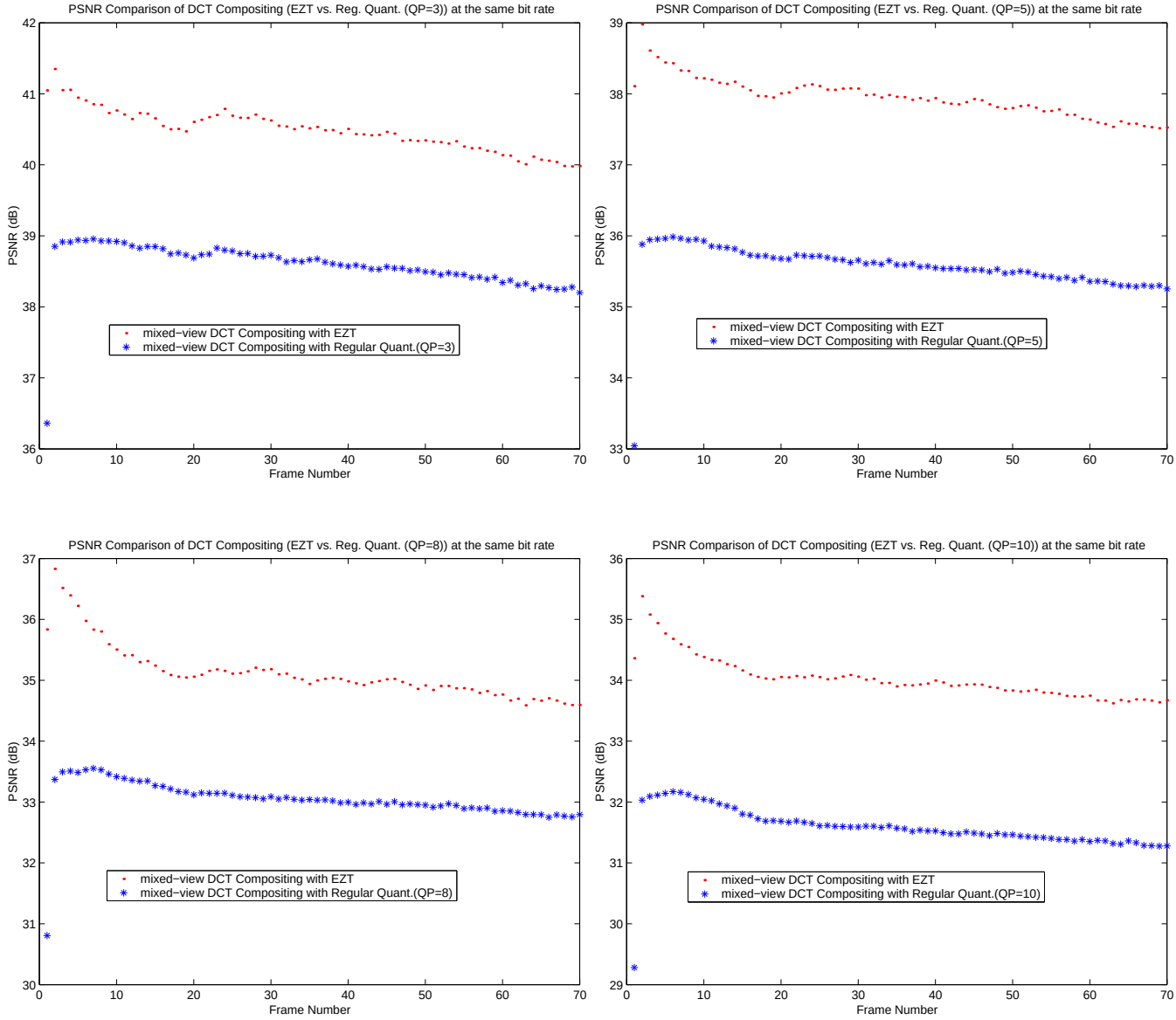


Figure 27: PSNR comparisons of DCT-EZT and conventional DCT encoder for mixed-view compositing

(a) Original composited frame (Frame # 5)

(b) Conventional DCT (QP=3), 93174 bits
PSNR=38.9412 dB

(c) DCT-EZT, 93174 bits
PSNR=40.9455 dB

Figure 28: Mixed-view composited video frame samples from the conventional DCT and the DCT-EZT codings

(d) Conventional DCT (QP=5), 64231 bits
PSNR=35.9628 dB

(e) DCT-EZT, 64231 bits
PSNR=38.4407 dB

(f) Conventional DCT (QP=10), 33959 bits
PSNR=32.1421 dB

(g) DCT-EZT, 33959 bits
PSNR=34.7676 dB

Figure 28 (Cont.): Mixed-view composited video frame samples from the conventional DCT and the DCT-EZT codings

We also compare our embedded zerotree coder with the other zerotree coders. In the first comparison, we show the PSNR results of our DCT-based embedded zerotree coder (DCT-EZT) for a frame from QCIF (Quadrature Common Intermediate Format) Foreman sequence in Fig. 29. The other PSNR values in the same figure is from a wavelet based embedded zerotree coder (EZW) based on Shapiro's work in [15]. As seen from the figure, we obtain better PSNR values. However it must be stated that in the wavelet based method there are 2- scales of wavelets while in our method we use 3-scale structure.

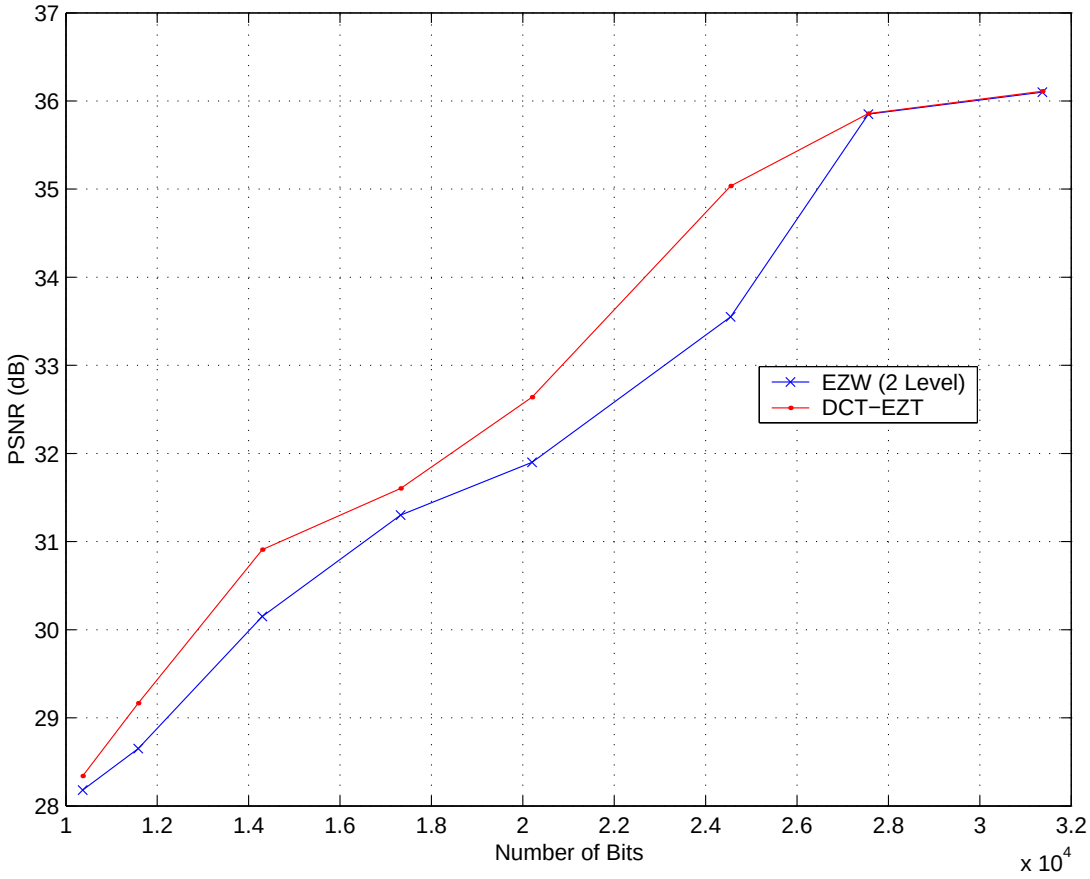


Figure 29: Comparison of DCT-EZT and Wavelet based embedded zerotree coder

We make another comparison with the results in [22] where 3-scale wavelet and additionally a virtual decomposition of the coarsest subband are used for Salesman CIF sequence. The results are shown in Fig. 30. Here our algorithm performed well enough so the results from both our algorithm and the one in [22] are very close to each other. However since

the other algorithm has enlarged zerotrees using virtual decomposition and uses a modified version of embedded zerotree coding it has slightly better results as seen from the figure.

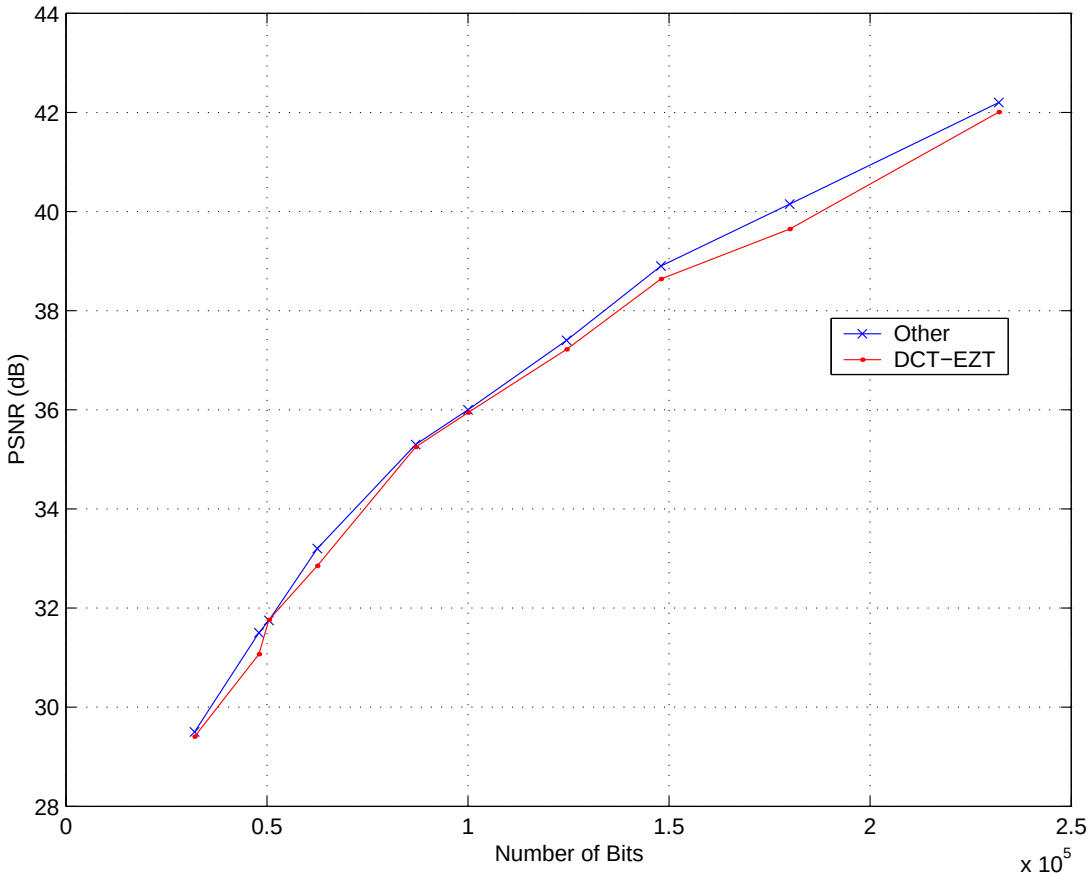


Figure 30: Comparison of DCT-EZT and embedded zerotree coder with virtual set partitioning in hierarchical tree

As a result we see that our embedded zerotree coding algorithm does well enough when comparing it to other zerotree coding algorithms. Beside it is also possible to improve our algorithm by using different strategies for obtaining zerotrees as stated in [21].

3.6 APPLICATION OF DCT DECIMATION/INTERPOLATION WITH EMBEDDED ZEROTREE CODING TO A SINGLE VIDEO STREAM

In this chapter, motivated by the compositing of several video sources in multi-point video conferencing we come up with a method for single video streams that uses the proposed DCT-decimation in the encoder and the corresponding interpolation in the decoder. Our codec performance is illustrated using different decimation factors, and showing that it is particularly efficient for low bit rates. We run our algorithms for different CIF video sequences. Here in the figures and tables, regular encoding stands for H.263 video coding, but the difference is we use embedded zerotree coding instead of regular scalar quantization. Consequently adaptive arithmetic coding is used after zerotree coding. This way we use the same encoding methodology to encode the DCT coefficients and the data symbols with the proposed system with DCT decimation and interpolation. We also use full pixel motion estimation and compensation in both systems. So the comparison results are fair. Our system is shown in Fig. 31.

Error frames can be decimated by different decimation factors. We investigate the effect of decimation factor to the reconstructed image quality for given bit rates. We also use only $q \times q$, $(q \leq 8)$, part of the 8×8 DCT blocks as before in compositing. So the number of computations required to decimate an image is decreased.

We use integer and rational decimation factors with our proposed video coding system. The effects of each decimation factor differ with the number of bits used to encode each frame. For lower bit rates we obtain better PSNRs for an interpolated intrafame from a video sequence as shown in Fig. 32. Starting from the lowest bit rate, coding with the highest decimation factor obtains better results than the others since it has the smallest decimated video frame. However as the number of bits increases the efficiency of the high decimation factor decreases. This is because decimated frames requires only small number of bits to be encoded. So after a certain number of bits, decimated video frame will no longer require more bits and extra bits be wasted. Therefore the PSNR of the reconstructed video frame will saturate. As shown in Fig. 32 these PSNR saturations approximately occur after PSNR curves of regular and the proposed codings coincide at a particular bit rate. Thus the

maximum bit rate for an intraframe for a particular decimation factor can be accepted this coincidence point. For a GOP including an intraframe and interframes, PSNR comparisons of the proposed method with decimation factor, $N = 2$, and regular encoding are shown in Fig. 33. As seen from this figure, average PSNRs of a video sequence strongly depends on the bit rate of the intraframe. If intraframe is encoded with a low bit rate, which is our aim in this work, the proposed method gives better results than regular encoding. In Fig. 33 from left to right intraframes are encoded with 10000, 15000, and 20000 bits. In this example, we encode 30 frames/second and each GOP (Group of Picture) includes an intraframe and 49 interframes. For the video coding standart H.263, number of interframes can be up to 131 in a GOP. For example in the Salesman sequence, on the graphs on the left hand side and in the middle, our method gives better PSNRs from 7.47 kilobits/second up to 45 kilobits/second. However, we obtain worse results for the case where intraframe is encoded with 20000 bits as shown on the graph on the right hand side.

Reconstructed video frame samples are shown in Fig. 34 for regular and the proposed encoding with the decimation factor $N = 2$. Beside the objective results, the reconstructed frames are also subjectively better than the ones from regular coding. Blocking effects are especially more visible in intraframe-coded Salesman and Foreman video frames for regular encoding as seen in Fig. 34. However for the proposed encoding there is no disturbing blocking effects. In the proposed method, since we lose some high frequency information because of the decimation, some blurring occurs on the detailed part of the images like books on the shelves in Salesman frame. Nevertheless for video conferencing sequences background information is not important since the viewers are only interested in people talking.

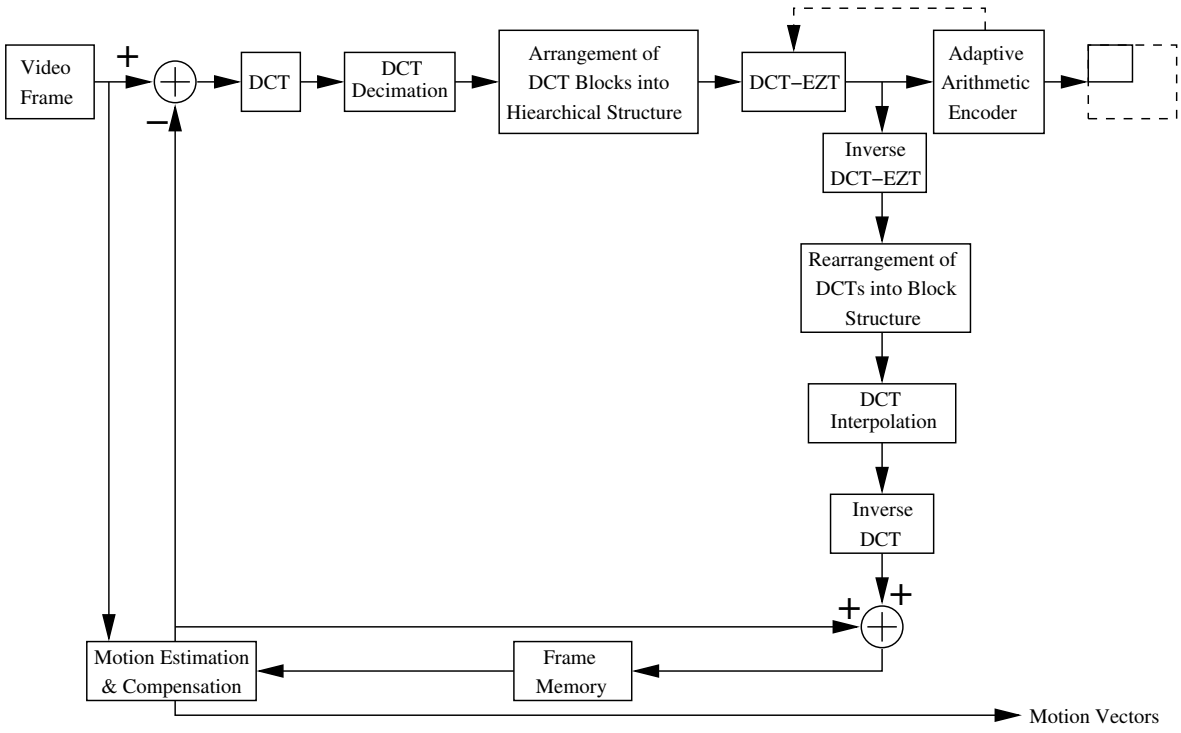


Figure 31: Proposed encoder

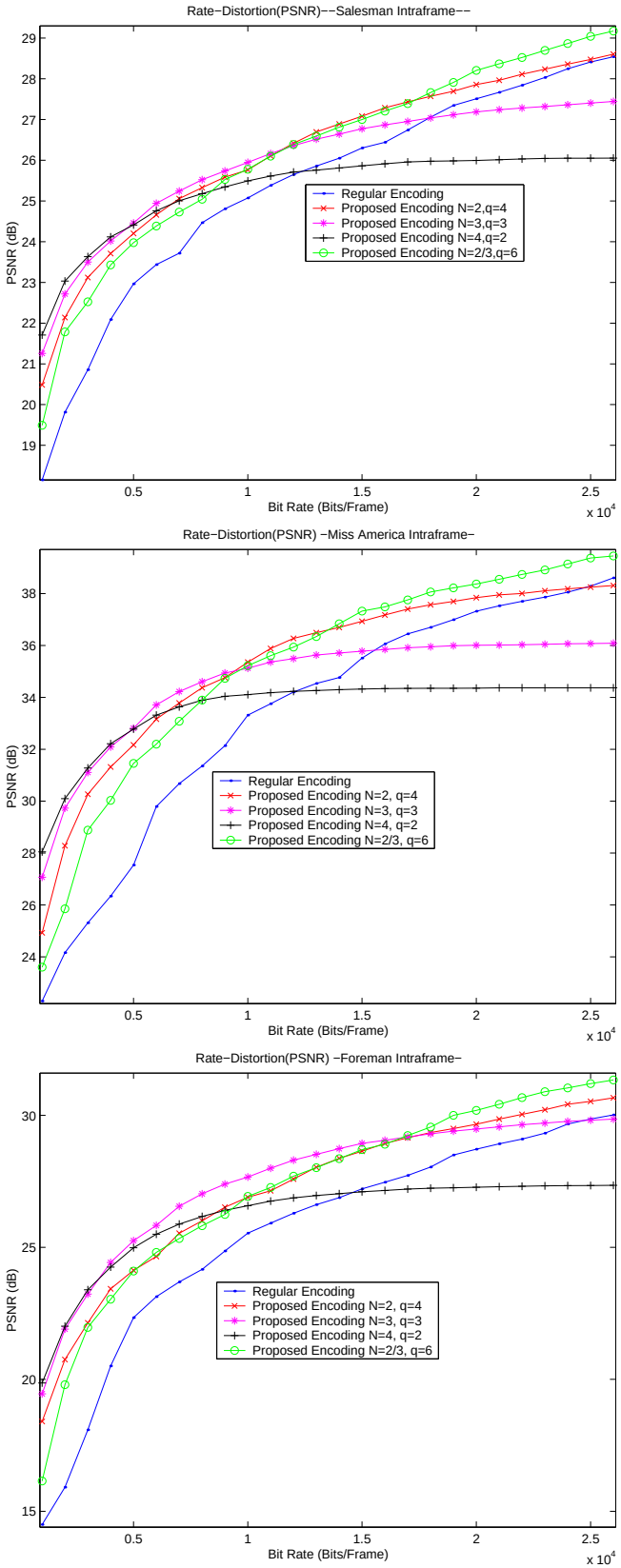
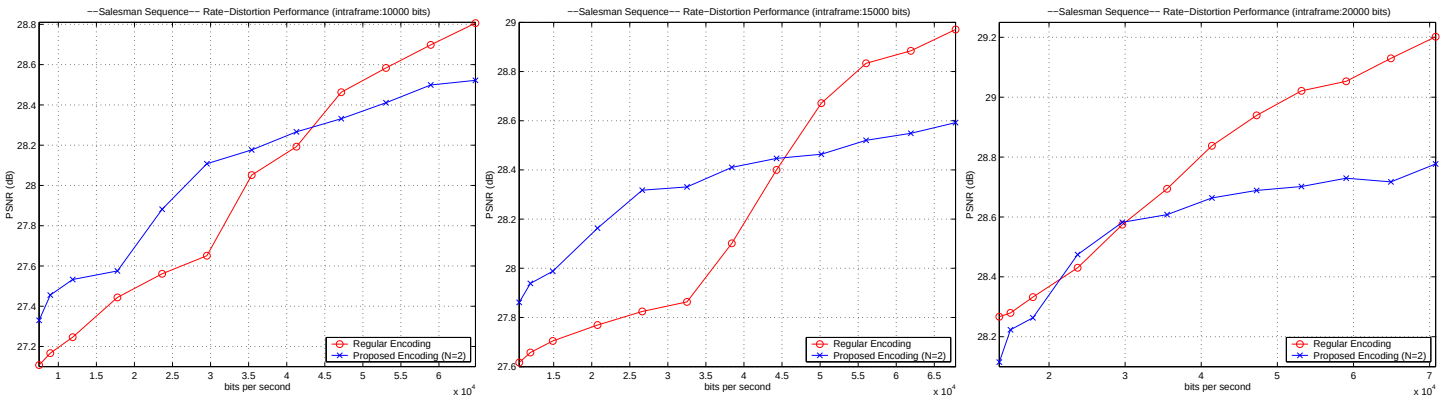
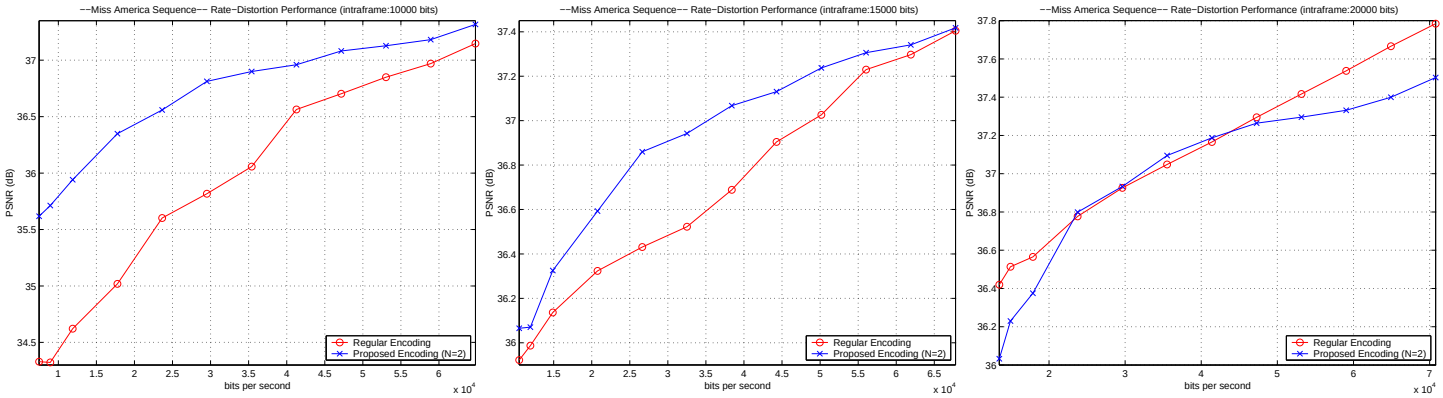


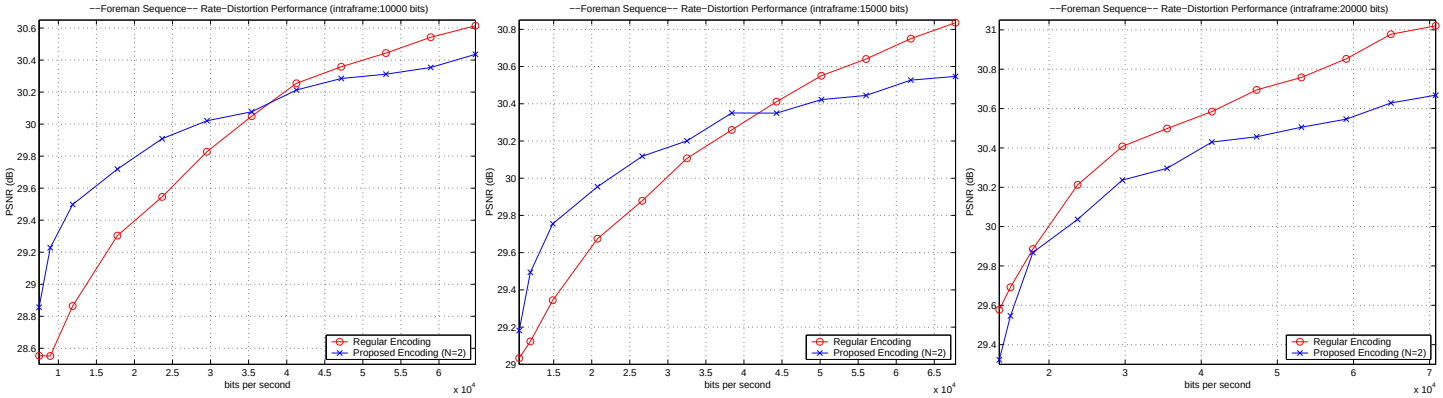
Figure 32: Rate-distortion performances of the proposed encoding vs. regular encoding for intraframes (from top to bottom: Salesman, Miss America, and Foreman)



(a) Salesman Sequence

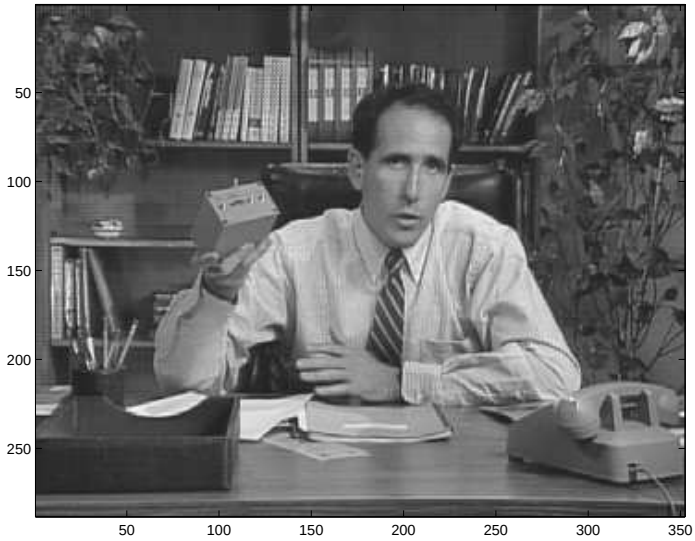


(b) Miss America Sequence

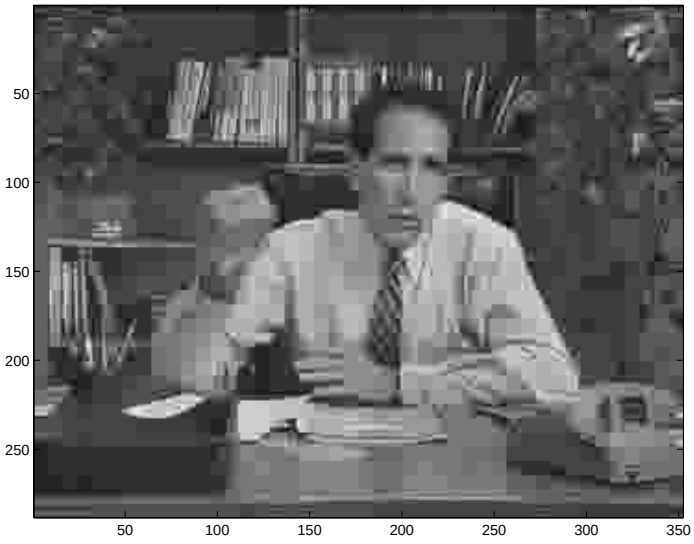


(c) Foreman Sequence

Figure 33: Rate-distortion performances of the proposed encoding vs. regular encoding for interframes



(a) Original Salesman Frame (Frame # 1)



(b) Regular, 0.5 bits/pixel
PSNR=27.56 dB

(c) Proposed, 0.5 bits/pixel
PSNR=27.88 dB

Figure 34: Video frame samples from regular and proposed codings

(d) Original Miss America Frame (Frame # 1)

(e) Regular, 0.5 bits/pixel
PSNR=37.40 dB

(f) Proposed, 0.5 bits/pixel
PSNR=37.88 dB

Figure 34 (Cont.): Video frame samples from regular and proposed codings

(g) Original Foreman Frame (Frame # 1)

(h) Regular, 0.5 bits/pixel
PSNR=28.79 dB

(i) Proposed, 0.5 bits/pixel
PSNR=29.74 dB

Figure 34 (Cont.): Video frame samples from regular and proposed codings

4.0 IMPROVEMENT OF ZEROTREE CODING

In this chapter, we investigate another approach of zetotree coding to improve the coding efficiency. Details of this approach and comparison with the zerotree coding method explained in Chapter 3 is given in the following two sections. Another improvement is achieved by application of local cosine transform (LCT) at low bit rates. This method is used together with zerotree coding to decrease the blocking effect, which is visible at low bit rate coding. Theory of this method and comparison with the DCT-based zerotree coding is given in the second section of the chapter.

4.1 SET PARTITIONING IN HIERARCHICAL TREES

Besides embedded zerotree coding being superior to regular scalar quantization based coding, it is still possible to improve this method further by using the approach in [21], originally derived for wavelet. This method is called set partitioning in hierarchical trees (SPIHT) and is performed in two passes: sorting pass and refinement pass. Different from the original zerotree coder, SPIHT obtains completely binary symbols at the output. Thus using only a single alphabet with two symbols increases the performance of the adaptive arithmetic coder.

To implement the method, DCT coefficients in 8×8 blocks are first rearranged into the subband structure as stated in Section 3.2. There are three lists maintained in the SPIHT algorithm: list of insignificant coefficients (LIC), list of significant coefficients (LSC), and list of insignificant sets (LIS). At the initial stage, LIC contains all the coefficients in the highest subband, and LIS includes the set of descendants of each such coefficient. LSC is initially

empty. In the LIS, descendants are categorized in two types as *Type A* and *Type B*: The set of all descendants of node (i, j) is labeled as $D(i, j)$ or *Type A*. Offsprings (direct descendants) of node (i, j) is identified as $O(i, j)$. So $L(i, j) = D(i, j) - O(i, j)$ is descendants of node (i, j) except the direct ones and labeled as *Type B*. Initially all nodes with descendants are added to the LIS as *Type A* entries. For a given threshold, the relationship between magnitude comparisons and outputted bits can be shown as follows:

$$S_{Th}(SET) = \begin{cases} 1, & max\{|C_{i,j}|\} \geq Th \\ 0, & otherwise \end{cases} \tag{4.1}$$

Here SET is either $D(i, j)$ or $L(i, j)$. In the same manner, significance of a coefficient $C_{i,j}$ can be defined such that $S_{Th}(C_{i,j}) = 1$, if $|C_{i,j}| \geq Th$; $S_{Th}(C_{i,j}) = 0$, otherwise. During the sorting pass if a set is found to be significant it is partitioned into subsets as will be explained. The objective is to obtain new partitions such that subsets expected to be insignificant contain a large number of coefficients, and subsets expected to be significant contain only one element [21]. This way the number of magnitude comparisons and therefore number of outputted bits is reduced.

The sorting pass is made for LIC and LIS. The algorithm works its way down the LIC first, comparing magnitudes of coefficients with the current threshold. If a coefficient is found to be significant, first a "1" bit for its significance, and then according to its sign "0" for negative or "1" for positive are outputted, and the coefficient is moved to the LSC. If the coefficient is insignificant a "0" bit is sent. After testing the LIC, the LIS is tested. During the LIS test, as stated in Eq. 4.1, if all the descendants in a set are insignificant, it is indicated by just outputting one bit, "0". If a set contains at least one significant coefficient, a "1" is sent to decoder and the set is partitioned into subsets. If the set of (i, j) node is of *Type A* then $D(i, j)$ is partitioned into $L(i, j)$ and four single coefficients $C_{k,l} \, \epsilon \, O(i, j)$. Then node (i, j) is moved to the end of the LIS as of *Type B* to be compared with the current threshold after completing the comparison of the sets in turn. Later each of four $C_{k,l} \, \epsilon \, O(i, j)$ is tested for significance. If one is found to be significant a "1" and corresponding bit for its sign are outputted, and the coefficient is added to the LSC. If the coefficient is insignificant, it is added to the end of the LIC and a "0" is sent to decoder. If the set of (i, j) node

is of *Type B* and is significant, first a "1" is outputted and $L(i,j)$ is partitioned into four sets $D(k,l)$ with $(k,l)\ \epsilon\ O(i,j)$. Then each $(k,l)\ \epsilon\ O(i,j)$ is added to the end of the LIS as an entry of *Type A* and (i,j) is removed from the LIS to be compared with the current threshold after finishing the comparison of the sets in turn. An example of partitioning *Type A* or *Type B* entries is shown in Fig. 35.

After the sorting pass is completed, the refinement pass for LSC, which is the same with the subordinate pass explained in Chapter 3, is performed. After an entire pass is made, the threshold is halved. As seen the idea of the SPIHT in [21] is derived from the embedded zerotree coding in [15]. The crucial differences are the partitioning of the coefficients and how the significant information is conveyed to the decoder. The flowchart of the algorithm is given in Fig. 36. As in the other zerotree coding each symbol is encoded by an adaptive arithmetic encoder as soon as it is outputted. The algorithm continues until the desired bit rate is reached.

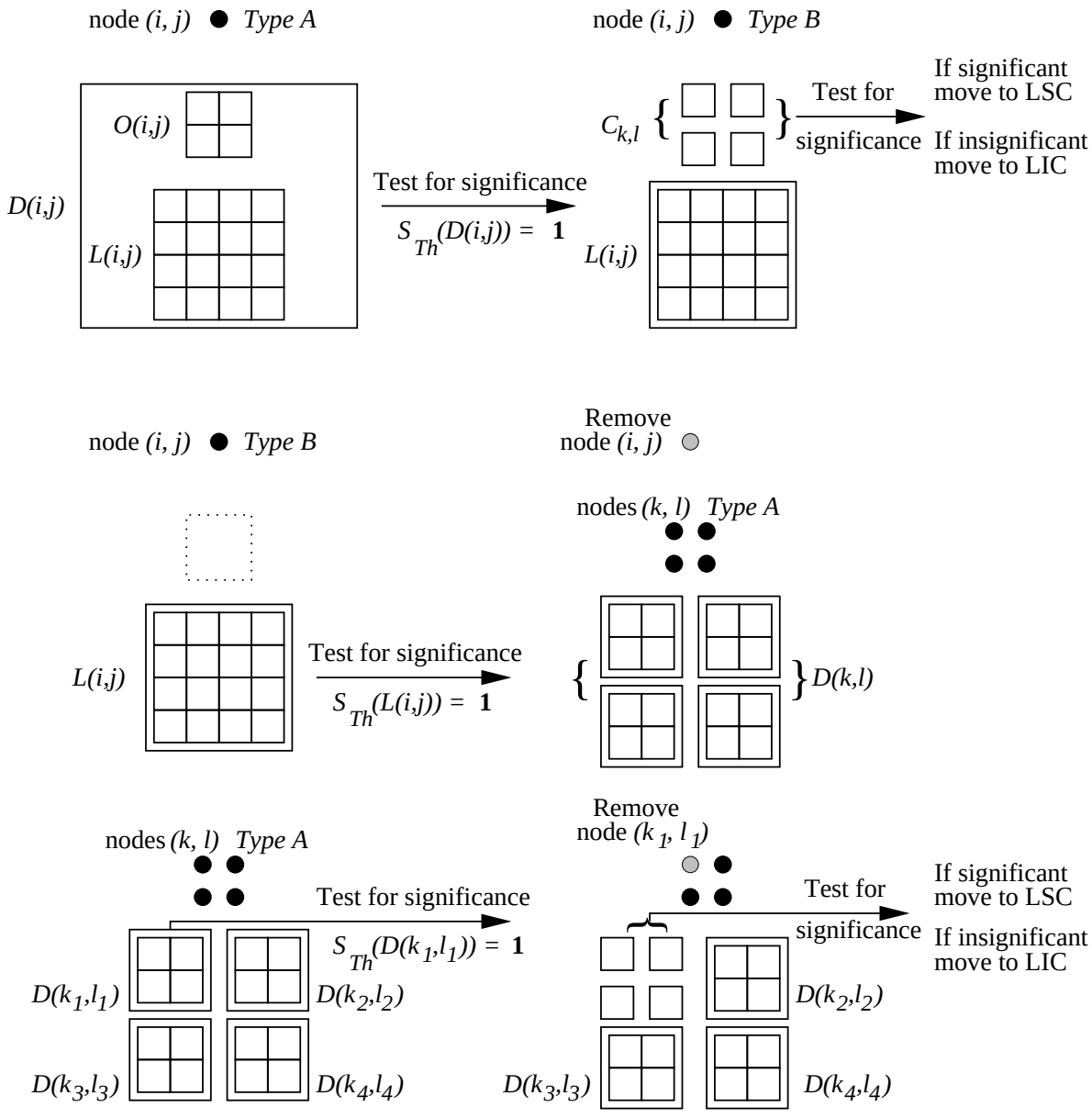


Figure 35: Set partitioning examples

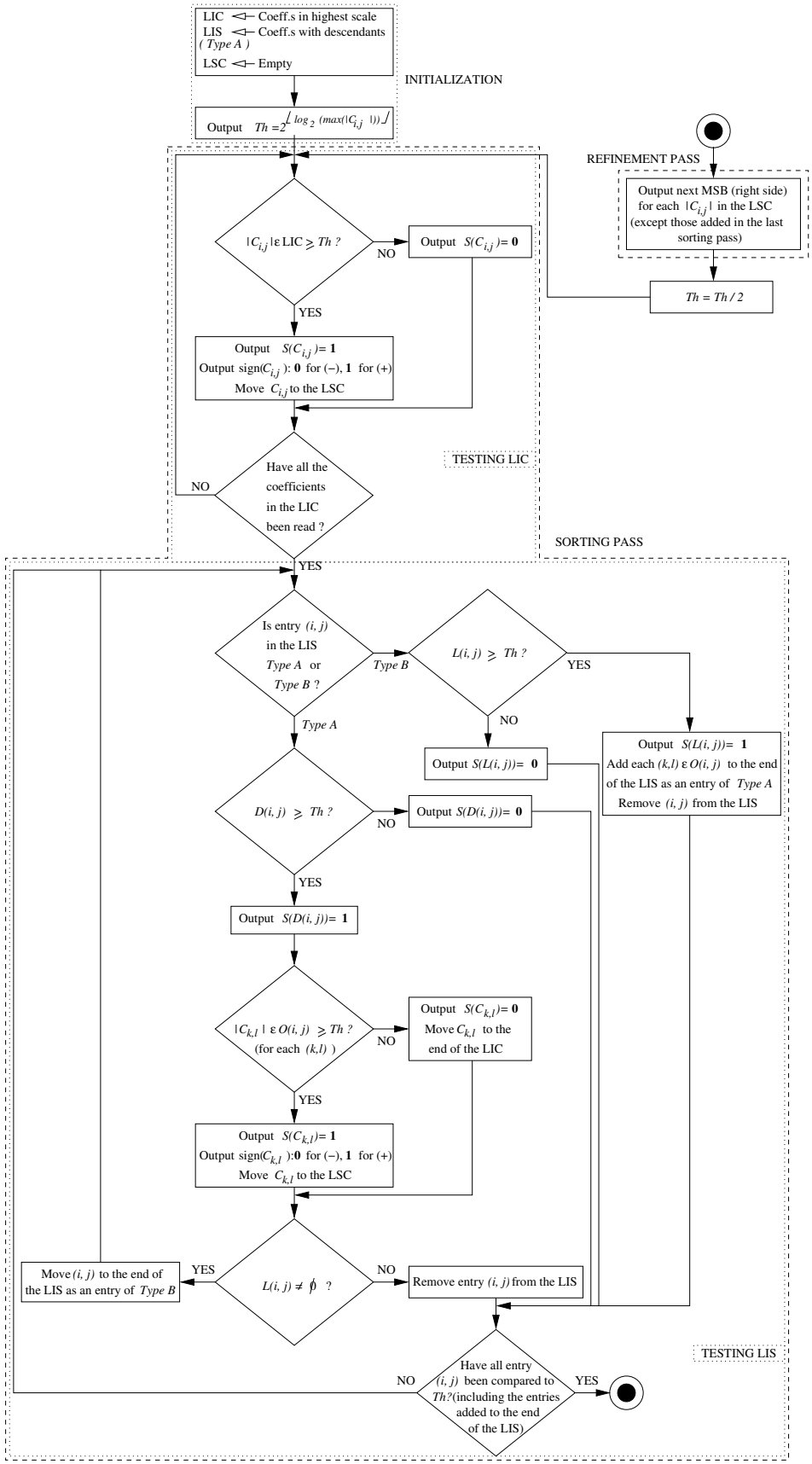


Figure 36: Flowchart of SPIHT

4.1.1 Comparison of DCT-EZT with DCT-SPIHT

In this section, we compare the improved embedded zerotree coding with the regular zerotree coding. Since the improved one is called SPIHT, we will call the DCT-based method DCT-SPIHT. For composited videos with four subframes the comparison results are shown in Table 8. Here average PSNR values for 50 frames are given at corresponding certain bit rates. The first frames are always encoded as intraframe at the bit rate of four times of the bit rate of interframes. We also show individual PSNR values for each frame of the composited video sequence at certain bit rates in Fig. 37. As seen from Table 8 and Fig. 37, we obtain higher PSNR values with DCT-SPIHT at each bit rate. Some reconstructed frame samples with four subframes are shown in Fig. 38 for both methods. As seen from these samples DCT-SPIHT also subjectively gives better results than DCT-EZT. It is especially visible in Salesman subframe, which is the most detailed one among four subframes.

For composited videos with six subframes, the average PSNR values of the DCT-SPIHT and DCT-EZT are given in Table 9. In this case, DCT-SPIHT gives higher PSNRs than DCT-EZT does at most bit rates except 20000 bits/frame. At this bit rate, we obtain slightly better results with DCT-EZT. In Fig. 39, PSNR values of each frame at certain bit rates are shown. We display some video frame samples with six subframes from both methods in Fig. 40. As seen from the composited video frame samples, quality of the reconstructed video frames from DCT-SPIHT is visually better than those from DCT-EZT.

Table 8: Average PSNR comparisons of DCT-EZT and DCT-SPIHT for composited videos with four subframes

Bit Rate (bits/interframe)	DCT-EZT	DCT-SPIHT
2500	27.0877 dB	27.7065 dB
5000	28.6256 dB	29.2535 dB
10000	31.6341 dB	32.1707 dB
15000	33.4862 dB	34.0600 dB
20000	35.3069 dB	35.6364 dB
25000	35.7594 dB	36.3797 dB

Table 9: Average PSNR comparisons of DCT-EZT and DCT-SPIHT for composited videos with six subframes

Bit Rate (bits/interframe)	DCT-EZT	DCT-SPIHT
2500	25.0518 dB	25.6902 dB
5000	27.8878 dB	28.3520 dB
10000	31.0797 dB	31.1779 dB
15000	31.8695 dB	32.4798 dB
20000	34.9613 dB	34.8213 dB
25000	35.6623 dB	36.0816 dB

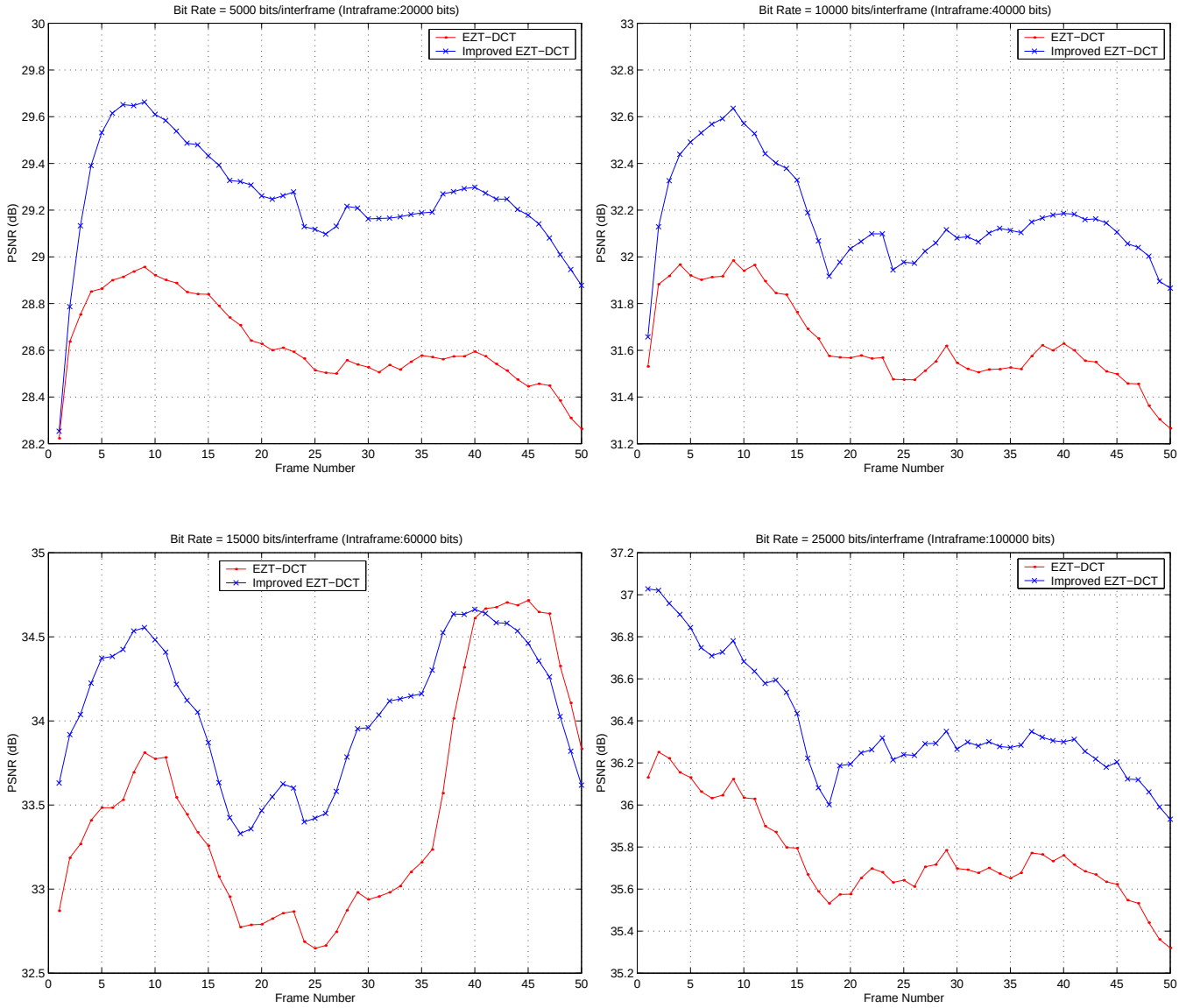


Figure 37: PSNR comparisons of DCT-EZT vs. DCT-SPIHT for composited videos with four subframes

(a) Original composited frame (Frame # 10)

(b) DCT-EZT, 5000 bits

PSNR=28.9220 dB

(c) DCT-SPIHT, 5000 bits

PSNR=29.6105 dB

Figure 38: Composited video frame samples with four subframes from DCT-EZT and DCT-SPIHT

(d) DCT-EZT, 10000 bits

PSNR=31.9405 dB

(e) DCT-SPIHT, 10000 bits

PSNR=32.5713 dB

(f) DCT-EZT, 15000 bits

PSNR=33.7740 dB

(g) DCT-SPIHT, 15000 bits

PSNR=34.4821 dB

Figure 38 (Cont.): Composited video frame samples with four subframes from DCT-EZT and DCT-SPIHT

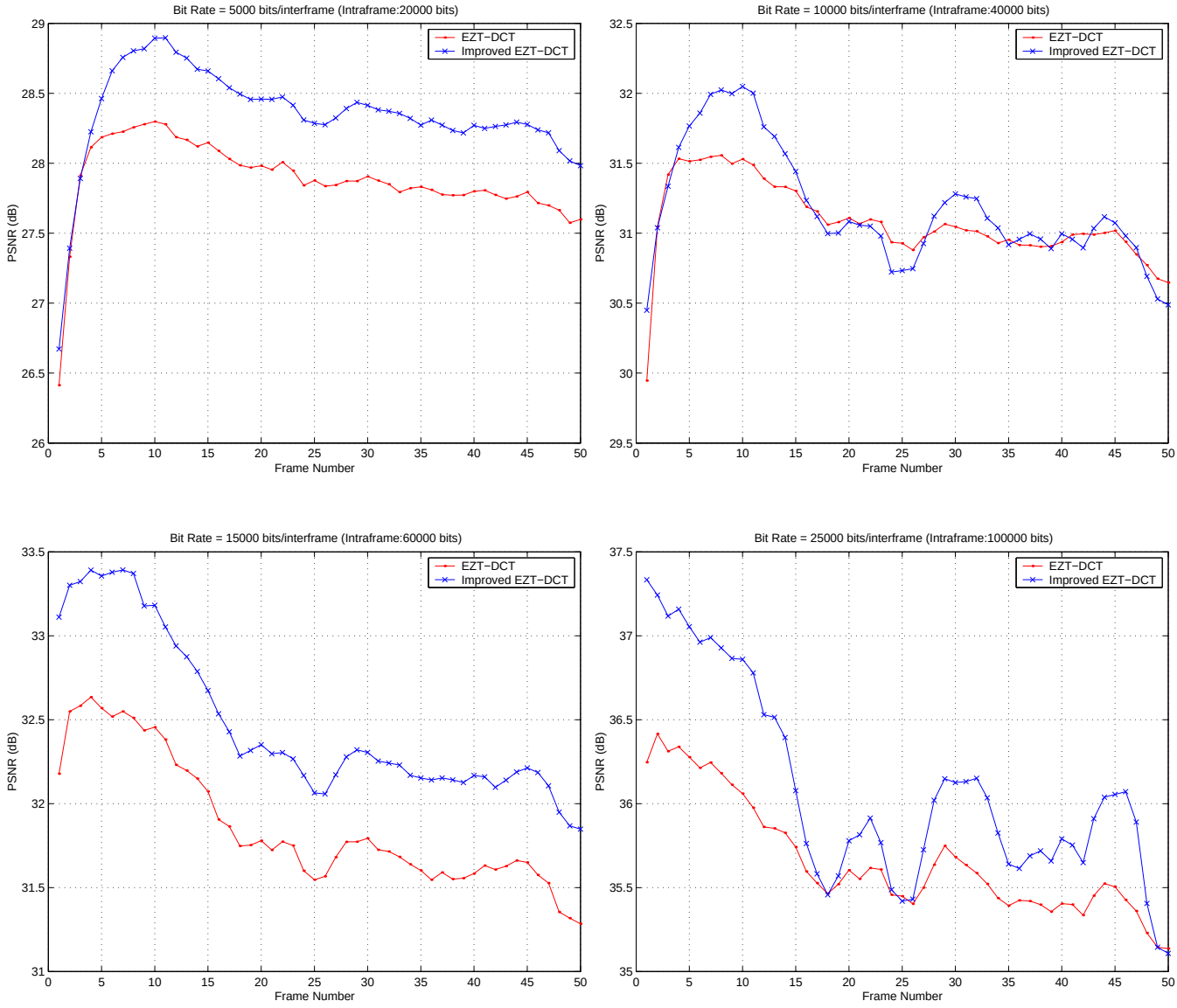


Figure 39: PSNR comparisons of DCT-EZT vs. DCT-SPIHT for composited videos with six subframes

(a) Original composited frame (Frame # 10)

(b) DCT-EZT, 5000 bits
PSNR=28.2981 dB

(c) DCT-SPIHT, 5000 bits
PSNR=28.8940 dB

Figure 40: Composited video frame samples with six subframes from DCT-EZT and DCT-SPIHT

(d) DCT-EZT, 10000 bits

PSNR=31.5297 dB

(e) DCT-SPIHT, 10000 bits

PSNR=32.0487 dB

(f) DCT-EZT, 15000 bits

PSNR=32.4553 dB

(g) DCT-SPIHT, 15000 bits

PSNR=33.1812 dB

Figure 40 (Cont.): Composited video frame samples with six subframes from DCT-EZT and DCT-SPIHT

4.2 REDUCTION OF BLOCKING EFFECT AT LOW BIT RATES

At low bit rates, the main problem of the DCT-based image and video coding is the blocking effect in reconstructed images or video frames. This unwanted natural consequence is caused by the independent processing of each block. It is seen as discontinuities across block boundaries. Viewers can easily notice the blocking effect at low bit rate codings.

There is a method to reduce the blocking effect such as Lapped Orthogonal Transform (LOT) [34], [35], [36]. The optimal LOT is concerned with DCT-II in such a way that a fast LOT algorithm can be obtained [36]. However even a fast LOT algorithm requires 20-30% more computations than DCT-II does [37]. The other disadvantage of the LOT is that it was derived in order to replace DCT-II as the kernel of the transform coding [36], [37]. Therefore it is not convenient to be plugged into our method, since our method is completely based on DCT-II coding.

Another method to reduce the blocking effect is introduced with the idea that can be used with existing DCT-based encoders by applying a preprocessing stage to the source images or video frames directly [37]. Accordingly a postprocessing stage is added to the decoder to obtain the reconstructed images or video frames after inverse DCT is taken. This method is called Local Cosine Transform (LCT) [37]. In this method, transform bases are formed of a cosine or sine multiplied by a smooth bell (cutoff) function that overlaps contiguous blocks. Consequently discontinuities across the block boundaries are reduced and smoothed.

As seen from Fig. 41, the preprocessing stage is the folding process, which is the application of the bell function to the blocks of the error frame in the spatial domain at the encoder. Then DCT is applied to each block. These two processes are together called LCT. At the decoder, inverse LCT includes the inverse DCT and unfolding operations.

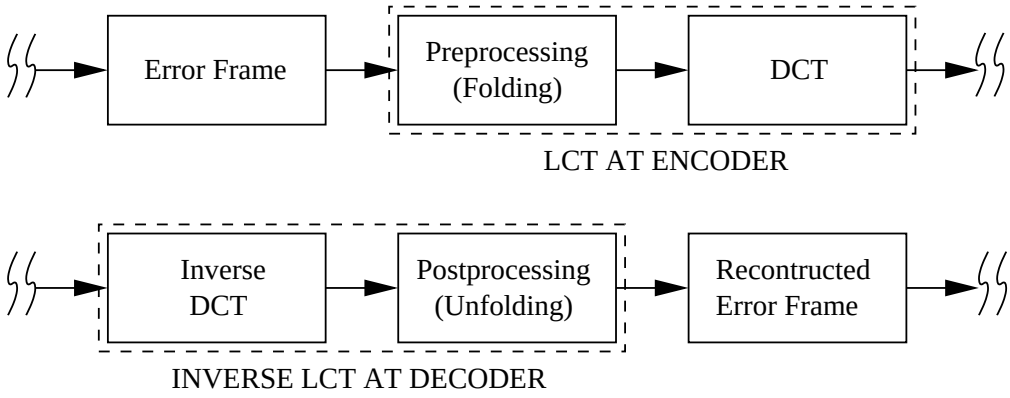


Figure 41: Local cosine transform

4.2.1 Theory of Local Cosine Transform

In LCT, bell function is used to fold adjacent blocks to each other. Properties of one dimensional bell functions $b_j(x)$ defined over the intervals $I_j = [m_j, m_{j+1}]$ are given [37] as

$$\begin{cases} 0 \leq b_j(x) \leq 1, \quad for\ all\ x, \\ b_j(x) = 1, \quad x \in [m_j + \varepsilon_j, m_{j+1} - \varepsilon_j], \\ b_j(x) = 0, \quad x \notin [m_j - \varepsilon_j, m_{j+1} + \varepsilon_{j+1}], \\ b_{j-1}(x) = b_j(2m_j - x)\ and\ b^2(x) + b^2_{j-1}(x) = 1, \quad x \in [m_j - \varepsilon_j, m_j + \varepsilon_j]. \end{cases} \tag{4.2}$$

Eq. 4.2 states that the two bells $b_{j-1}(x)$ and $b_j(x)$, which are supported over the consecutive intervals I_{j-1} and I_j, are orthogonal and have a mutual symmetry with respect to m_j as shown in Fig. 42.

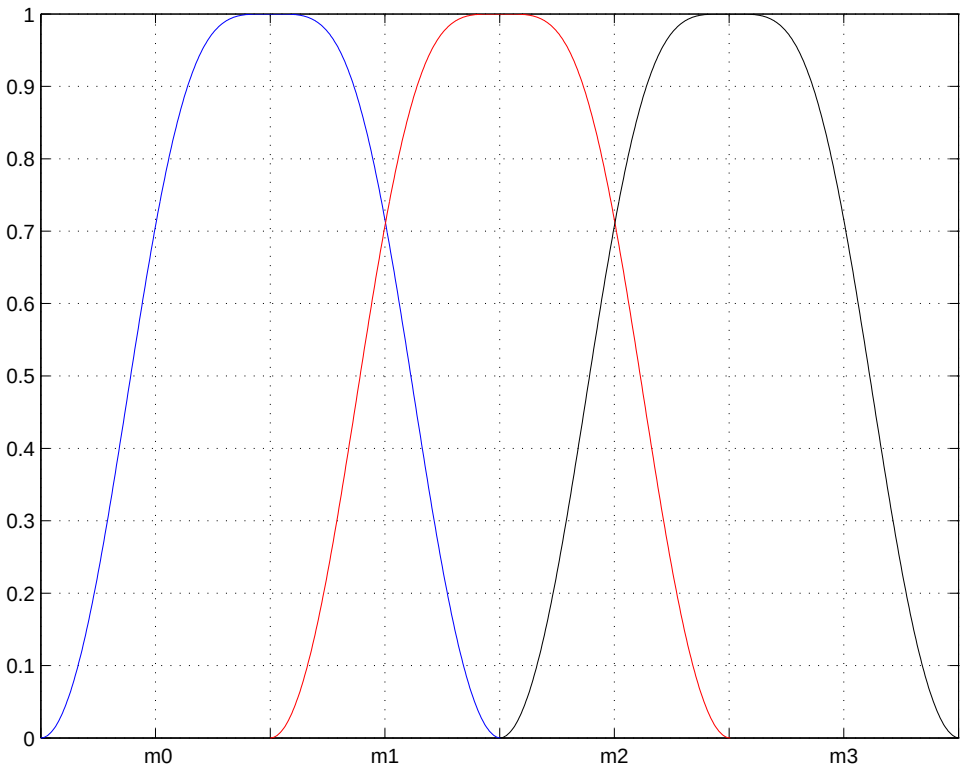


Figure 42: Consecutive intervals and corresponding bells

Let us now consider a one dimensional bell function $b(x)$ based on the function β that is given as

$$\beta(x) = \begin{cases} 0, & \text{if } x < -1, \\ \frac{1}{2}(1 + sin(\frac{\pi}{2}x)), & \text{if } -1 \leq x \leq 1, \\ 1, & \text{if } x > 1 \end{cases} \tag{4.3}$$

To implement LCT on video frames, frames are divided into blocks of size $N \times N$, where N is an integer. Then a one dimensional discrete symmetric bell of size of $2N$ is centered in the middle of each column and row block. By moving the bell from a block to its neighbor the two contiguous bells are overlapped by N pixels. Folding equation is given as

$$\begin{aligned} f_-(x) &= \frac{b(x)f(-x) - b(-x)f(x)}{b(x) - b(-x)}, \quad x \in [-\varepsilon_j, 0], \\ f_+(x) &= \frac{b(x)f(x) - b(-x)f(-x)}{b(x) - b(-x)}, \quad x \in (0, \varepsilon_j] \end{aligned} \tag{4.4}$$

where x is a continuous variable in both intervals $[-\varepsilon_j, 0]$ and $(0, \varepsilon_j]$. Here $f_-(x)$ and $f_+(x)$ are left and right folded functions within the intervals $[-\varepsilon_j, 0]$ and $(0, \varepsilon_j]$, respectively. As seen from this equation, function $f(x)$ is folded across 0 onto the intervals $[-\varepsilon_j, 0]$ and $(0, \varepsilon_j]$ by using the bell $b(x)$, and since the bell is symmetric, also by using $b(-x)$. Symmetry property of the bell function $b(x)$ is shown in Fig. 43. Original function $f(x)$ is reconstructed by using the unfolding equation given as follows:

$$f(x) = \begin{cases} b(x)f_+(-x) + b(-x)f_-(x), & x \in [-\varepsilon_j, 0], \\ b(x)f_+(x) + b(-x)f_-(-x), & x \in (0, \varepsilon_j]. \end{cases} \tag{4.5}$$

Instead of using the bell of size $2N$, a discrete bell of size N can be used as given [37]:

$$b(n) = \beta\left(\frac{n+1/2}{N/2}\right), \tag{4.6}$$

where $n = -N/2, -N/2 + 1, ..., N/2 - 1$, and β is the function defined in Eq. 4.3. In our case since we use 8×8 blocks ($N = 8$), particularly the values of the bell $b(n)$ for n =-4,

-3,...,3, 4 are the values of the function $\beta(x)$ for x =-7/8, -5/8, -3/8, -1/8, 1/8, 3/8, 5/8, 7/8. Therefore the folding equation will be obtained as follows:

$$f_-(n) = \frac{b(n)f(-n) - b(-n)f(n)}{b(n) - b(-n)}, \quad n = -4, -3, -2, -1, \tag{4.7}$$
$$f_+(n) = \frac{b(n)f(n) - b(-n)f(-n)}{b(n) - b(-n)}, \quad n = 0,\ 1,\ 2,\ 3.$$

In the same manner, unfolding will be applied as

$$f(n) = \begin{cases} b(n)f_+(-n) + b(-n)f_-(n), & n = -4, -3, -2, -1, \\ b(n)f_+(n) + b(-n)f_-(-n), & n = 0,\ 1,\ 2,\ 3. \end{cases} \tag{4.8}$$

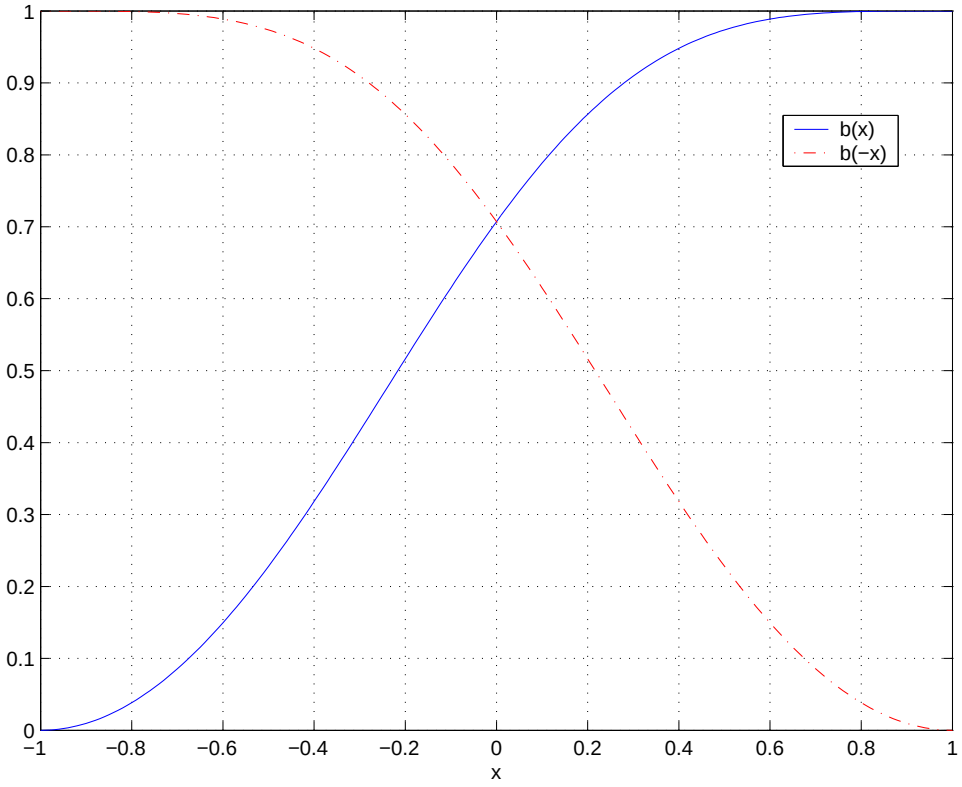


Figure 43: Symmetry property of bell function

In the folding operation the bell is positioned in the center of two contiguous blocks. For example, for two vertically contiguous blocks, the four pixels of the left block will be folded into the four pixels of the right block at the same row. In the same manner, four pixels of the

right block will be folded into the left block. The bell is moved across the boundary of the two contiguous blocks to fold other pixels of each block to the other's. The same operations are also applied to the horizontally contiguous blocks. After applying the folding operations to the all blocks of the image, forward DCT is applied to the resulting folded blocks.

The bell function defined on the basis of the function $\beta(x)$ may affect the reconstructed image quality (or the compression ratio). In order to obtain the optimal bell function an iterative method is used empirically in [37]. In this method the function β is given as

$$\beta(x) = \begin{cases} 0, & \text{if } x < -1, \\ \frac{1}{2}(1+z), & \text{if } -1 \leq x \leq 1, \\ 1, & \text{if } x > 1 \end{cases} \tag{4.9}$$

where z is described as follows:

$$\begin{aligned} &begin \\ &z = x \\ &for\ i = 1\ to\ iternum \\ &z = sin(\tfrac{\pi}{2}z) \\ &end \end{aligned}$$

The optimal iternum is found to be 3 [37]. Thus the optimal bell is based on the function β

$$\beta(x) = \begin{cases} 0, & \text{if } x < -1, \\ \frac{1}{2}(1 + sin(\frac{\pi}{2}sin(\frac{\pi}{2}sin(\frac{\pi}{2})))), & \text{if } -1 \leq x \leq 1, \\ 1, & \text{if } x > 1. \end{cases} \tag{4.10}$$

As displayed on the Fig. 44, as internum increases the bell function becomes flatter. Namely for large values of iternum, left half side of the bell function gets closer to 0 while right half side of the bell gets closer to 1. If left half side and right half side are exactly 0 and 1 respectively, then folding operation does nothing. Physically, the smaller the iternum, the smoother the resulting image or vice versa. However since smoother images do not mean optimal quality, in our experiments, we use the bell function with iternum 3 to obtain the optimal quality with LCT. Also as shown in Table 10, the PSNR results of some reconstructed

intraframes coded with LCT at several bit rates verifies that the best bell function is obtained with iternum 3.

In Fig. 45, we show some reconstructed composited video frames from Table 10 coded with LCT using three different bell functions. As seen from the figure, bell function with iternum 1 gives the smoothest but the worst frames. The bell function with iternum 6 supplies the least folding, thus blocking effects are clearly visible in the reconstructed frames. The best frames are obtained with the bell function with iternum 3.

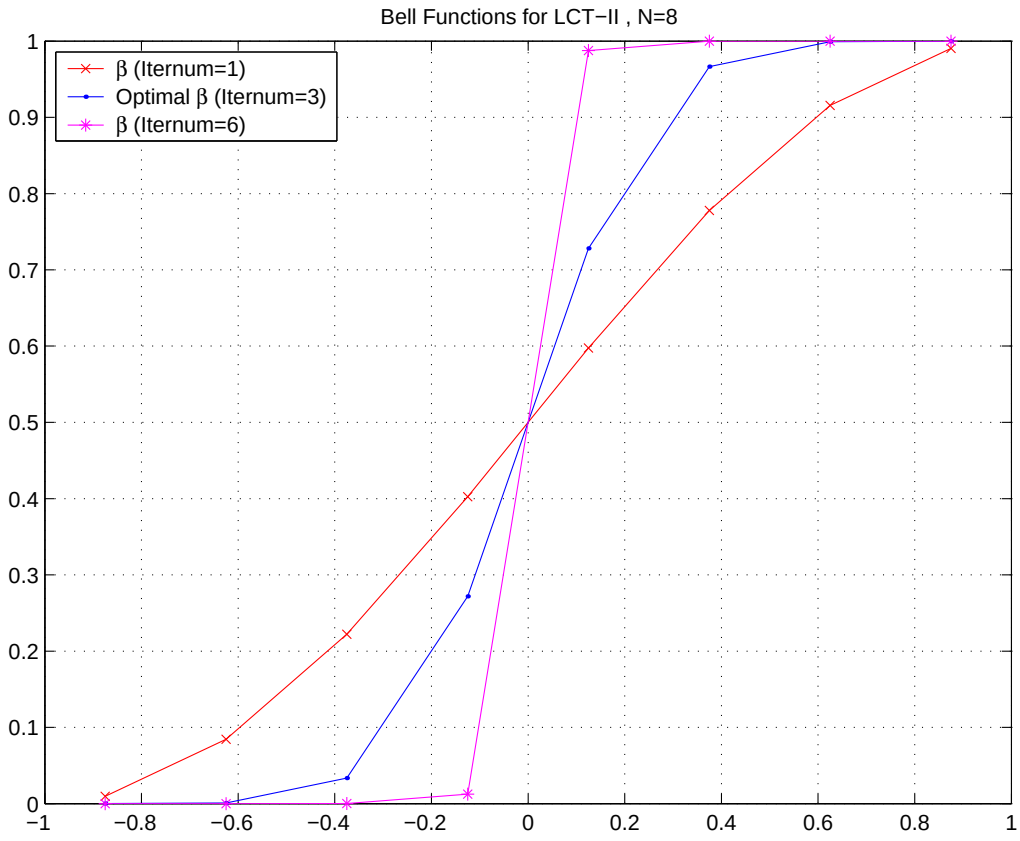


Figure 44: Bell functions for several iternums

Table 10: PSNR results of reconstructed frames coded with LCT with different bell functions

Video Frame	Bit Rate	β w. iternum=1	β w. iternum=3	β w. iternum=6
Miss America	15000 bits	35.2117 dB	36.4817 dB	35.6040 dB
	30000 bits	35.5935 dB	39.8523 dB	39.4684 dB
Salesman	15000 bits	25.7274 dB	26.7295 dB	26.3428 dB
	30000 bits	27.4089 dB	29.6320 dB	29.1936 dB
Foreman	15000 bits	26.9506 dB	28.0002 dB	27.2803 dB
	30000 bits	28.8500 dB	31.6424 dB	30.8868 dB
News	15000 bits	24.3859 dB	26.0539 dB	25.5081 dB
	30000 bits	28.2275 dB	30.4650 dB	29.6693 dB
Claire	15000 bits	30.3215 dB	32.0418 dB	31.2249 dB
	30000 bits	35.5232 dB	38.1987 dB	37.7871 dB
Hall	15000 bits	24.8939 dB	26.4207 dB	25.8662 dB
	30000 bits	29.0896 dB	31.1133 dB	30.2664 dB
Comp. (4-subfr.)	15000 bits	24.9371 dB	27.1013 dB	26.7383 dB
	30000 bits	27.3984 dB	30.1373 dB	29.9104 dB
Comp. (6-subfr.)	15000 bits	22.6630 dB	24.6238 dB	24.4627 dB
	30000 bits	24.8404 dB	27.6161 dB	27.5538 dB

Figure 45: Some video frame samples coded with LCT with different bell functions

4.2.2 Comparison of DCT-EZT with LCT-EZT

In this section, we compare the performance of the DCT-based embedded zerotree coding (DCT-EZT) with the LCT-based embedded zerotree coding (LCT-EZT) in our compositing system. To perform the LCT-EZT with the proposed compositing, we first need to obtain the LCT of the composited error frames. To do so, we take the inverse DCT of the error frames, then we apply LCT to the 8×8 blocks of the error frame. Then 8×8 LCT coefficients are arranged into hierarchical subband structure as explained before. The coefficients in hierarchical form are encoded by embedded zerotree coding. Resulting binary symbols are encoded by an adaptive arithmetic encoder. In the feedback loop, which is identical to the decoder, the coefficients are rearranged into the 8×8 block structure to apply inverse LCT. Then composited error frame is fed to the DCT operation. Obtained DCT error frame is added to the previous DCT-motion compensated frame, and the resulting frame is put in DCT frame memory. The proposed LCT-EZT encoder is shown in Fig. 46. Since we consider that the incoming video sequences are in the DCT domain we take the inverse and forward DCT in the compositing system at the appropriate places since the folding and unfolding are only applied in spatial domain. However, if incoming frames are in the LCT domain; DCT and inverse DCT are excluded resulting in decrease of number of computations in the proposed encoder.

We first implement LCT-EZT with composited frames with four subframes. Average PSNR of the reconstructed video frames are shown in Table 11. At lower bit rates LCT-EZT gives better PSNR values than DCT-EZT does. However, as bit rate increases performance of LCT-EZT decreases. The reason is that the blocking effect decreases with the increment of bit rate. Individual PSNR results for each reconstructed frames with four subframes are shown in Fig. 47. In the PSNR plots, the first values are for intraframes while the others are for interframes. As seen from this figure, decoded intraframes of LCT-EZT gives better PSNR values than DCT-EZT up to 60000 bits/intraframe. However interframe performance of LCT-EZT is worse than that of DCT-EZT even at 5000 bits/interframe coding bit rate. This implies that motion estimation/compensation helps to improve the quality of the interframes decreasing the objective performance of LCT. Although LCT-EZT gives fewer

PSNR values, subjectively it decreases the blocking effect so that the reconstructed frames look smoother (see Fig. 48 (e), (f)). In Fig. 48, there are some reconstructed video frame samples from both LCT-EZT and DCT-EZT coding.

We also apply LCT-EZT algorithm to the composited frames with six subframes. The average PSNR results for this case is shown in Table 12. PSNR values of each reconstructed frames are shown in Fig. 49. Some reconstructed frame samples are given in Fig. 50. As in the four subframes case, we achieve better PSNR values with LCT-EZT than with DCT-EZT at low bit rates.

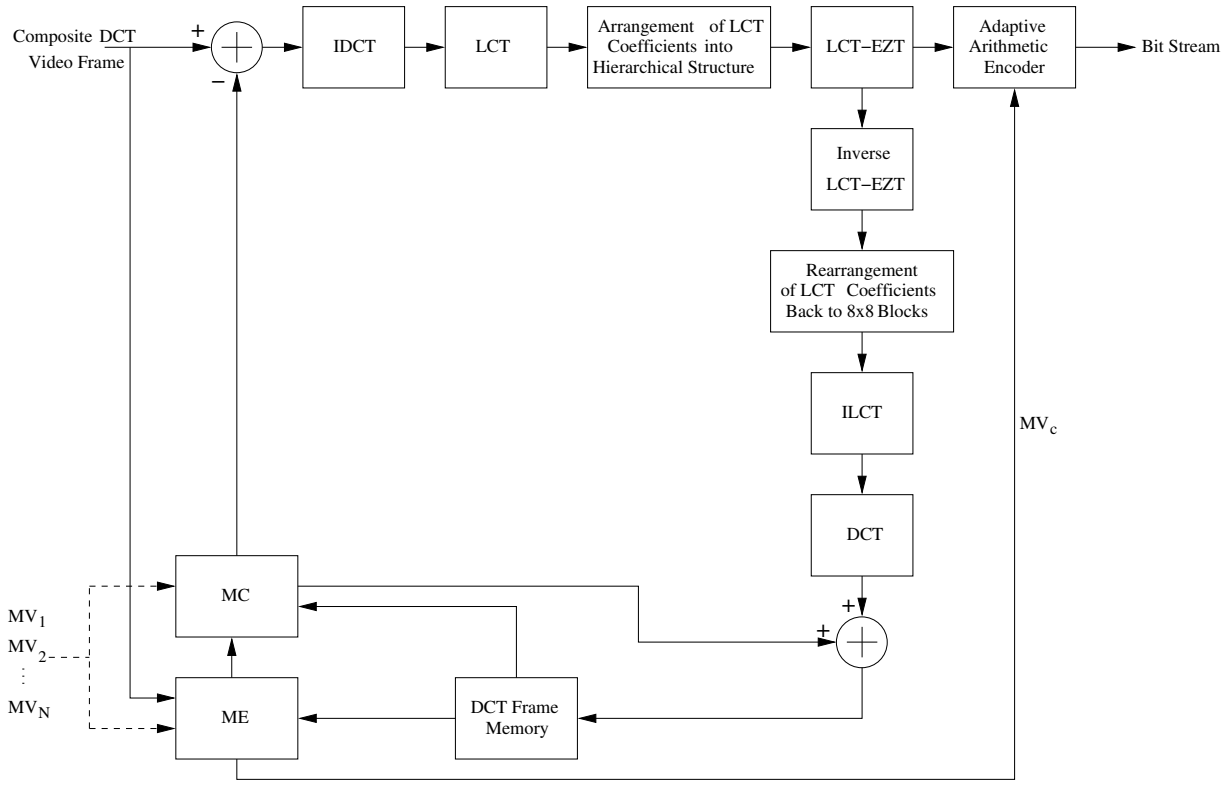


Figure 46: LCT-EZT encoder

Table 11: Average PSNR comparisons of LCT-EZT and DCT-EZT for composited videos with four subframes

Bit Rate (bits/interframe)	LCT-EZT	DCT-EZT
2500	25.9283 dB	25.5151 dB
5000	27.7735 dB	27.8819 dB
10000	29.5911 dB	30.5675 dB
15000	31.5364 dB	32.0396 dB
20000	32.6391 dB	33.5928 dB
25000	33.3722 dB	34.9117 dB

Table 12: Average PSNR comparisons of LCT-EZT and DCT-EZT for composited videos with six subframes

Bit Rate (bits/interframe)	LCT-EZT	DCT-EZT
2500	24.9675 dB	24.6405 dB
5000	26.0938 dB	26.6767 dB
10000	28.1851 dB	28.9139 dB
15000	29.5181 dB	30.4907 dB
20000	30.7313 dB	31.4579 dB
25000	31.5801 dB	32.9231 dB

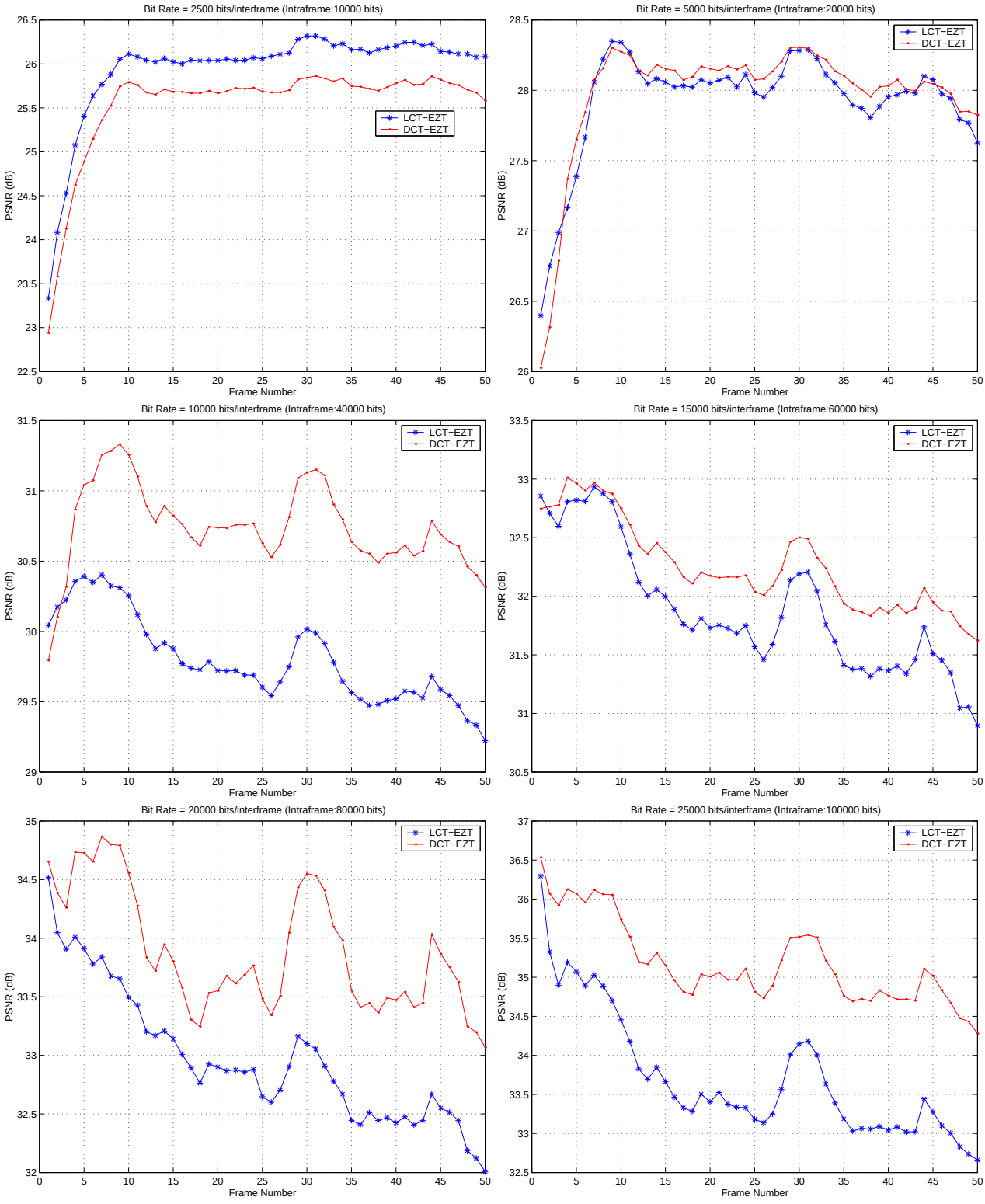


Figure 47: PSNR comparisons of composited frames with four subframes coded with LCT-EZT and DCT-EZT

(a) Original composited frame (Frame # 10)

(b) DCT-EZT, 2500 bits
PSNR=25.6832 dB

(c) LCT-EZT, 2500 bits
PSNR=26.0236 dB

Figure 48: Composited video frame samples with four subframes from DCT-EZT and LCT-EZT

(d) Original composited frame (Frame # 4)

(e) DCT-EZT, 5000 bits
PSNR=27.3693 dB

(f) LCT-EZT, 5000 bits
PSNR=27.1654 dB

Figure 48 (Cont.): Composited video frame samples with four subframes from DCT-EZT and LCT-EZT

(g) Original composited frame (Intraframe, Frame # 1)

(h) DCT-EZT, 40000 bits

PSNR=29.7942 dB

(i) LCT-EZT, 40000 bits

PSNR=30.0435 dB

Figure 48 (Cont.): Composited video frame samples with four subframes from DCT-EZT and LCT-EZT

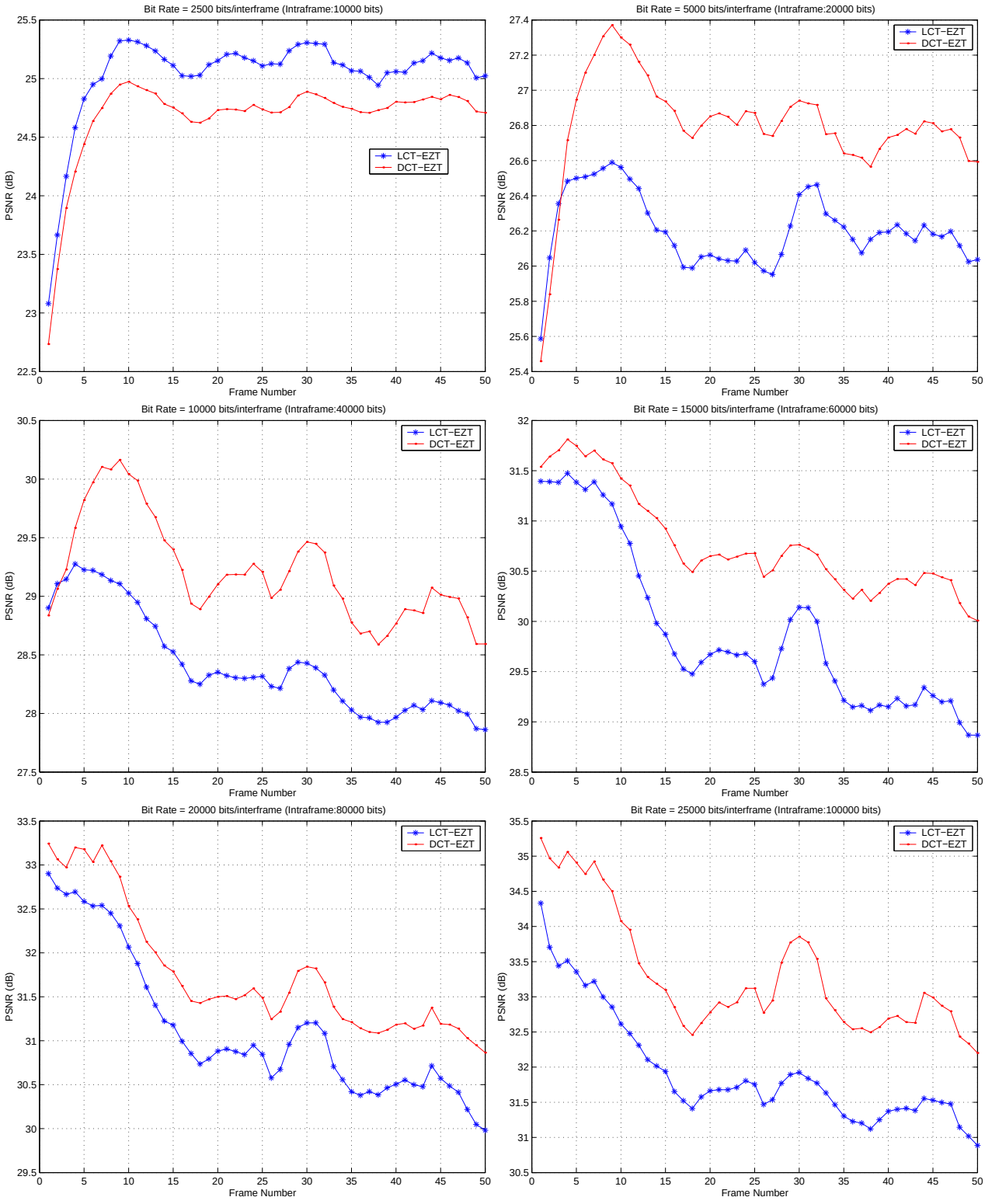


Figure 49: PSNR comparisons of composited frames with six subframes coded with LCT-EZT and DCT-EZT

(a) Original composited frame (Frame # 7)

(b) DCT-EZT, 2500 bits
PSNR=24.7484 dB

(c) LCT-EZT, 2500 bits
PSNR=24.9976 dB

Figure 50: Composited video frame samples with six subframes from DCT-EZT and LCT-EZT

(d) Original composited frame (Frame # 5)

(e) DCT-EZT, 5000 bits

PSNR=26.9467 dB

(f) LCT-EZT, 5000 bits

PSNR=26.4999 dB

Figure 50 (Cont.): Composited video frame samples with six subframes from DCT-EZT and LCT-EZT

(g) Original composited frame (Intraframe, Frame # 1)

(h) DCT-EZT, 40000 bits
PSNR=28.8369 dB

(i) LCT-EZT, 40000 bits
PSNR=28.9007 dB

Figure 50 (Cont.): Composited video frame samples with six subframes from DCT-EZT and LCT-EZT

5.0 BIT RATE CONTROL

In this chapter, we implement bit rate control for composited videos. Rate control problems can be generally characterized as the determination of the appropriate coding parameters by precoding and decoding processes so that the decoded video quality is optimized according to a certain bit rate.

Rate control for video compositing is generally different from rate control for the single video stream case. In such a case, the joint effect of each incoming video stream should be considered in the composited video. For instance, if one of the video streams contains more activity than the others, the number of bits assigned to this stream should be larger than those given to the others.

Depending on the channel conditions there are several bit rate control schemes for video coding. Most of them typically adopt a rate control scheme by adjusting the quantization step based on buffer occupancy. Some methods encode each image block several times with different quantization parameters (QP), and then select the best quantization parameter [38, 39]. However because of the high computational complexity these methods are not suitable for real-time applications [41]. Another method given in [40], selects the quantizers according to a formula derived from a model of the encoder. However, this approach does not achieve the exact target bit rate, and can suffer from frequent frame skipping and wasting of channel bandwidth in real time applications [41]. The embedded property of the zerotree coding greatly simplifies rate control since the coding control parameter is the allocated bit rate for each frame rather than the quantization parameter [42]. Additionally embedded zerotree coding gives better rate-distortion tradeoff while the encoded bit stream can be stopped at any point without a significant distortion [15, 20, 22, 42]. Thus we use the flexibility of embedded zerotree coding for bit rate control.

As shown in the previous chapters, the embedded property of the zerotree coding allows us to control the bit rate of each frame instantly. Therefore it is very easy to adapt the bit rate of a GOP to a given constant or variable channel bit rate. This can be basically done by allocating a fixed number of bits to each intraframe (I-frame) and interframe (P-frame). However, this scheme does not necessarily give the best average PSNR value for the decoded videos since it does not consider the rate-distortion performance of each frame. To improve the decoded video quality, the bit rate control problem can be formulated as a constrained optimization problem. This problem can be solved by Lagrangian method as will be explained later in this chapter.

5.1 RATE-DISTORTION MODEL AND RATE CONTROL PROBLEM

To solve the rate control problem, one needs to first obtain the rate-distortion model (R-D model) of a video frame. For this propose, each video frame is encoded and decoded at particular bit rates. Then one can easily find an approximation function for the R-D performance curve of each frame by using the obtained distortion versus bit rate graphics of decoded videos. Considering that R-D model of a frame is convex [42, 43], it can be formulated as

$$D = \sigma^2 2^{-\gamma R} \tag{5.1}$$

where D is the distortion, R is the bit rate, σ^2 is the variance of the DCT coefficients, and γ is the coding efficiency parameter. The variance σ^2 is also the coding distortion when bit rate R equals zero. This model can be easily verified by using experimental data for any video sequence. We show an example for convex R-D model of an I-frame from composited videos with four subframes in Fig. 51. The coding efficiency parameter γ specifies the decaying rate of the distortion as the bit rate increases. Generally coding efficiency parameter γ_I of I-frames are larger than the coding efficiency parameter γ_P of P-frames [42]. It is easy to see that the larger the coding efficiency parameter the more efficient the coding, because as bit rate increases distortion decays quickly with a higher coding efficiency parameter. We

show some R-D characteristics and coding efficiency parameters of some I and P frames in Fig. 52.

We know that a GOP has an I-frame followed by $N-1$ P-frames. If the channel capacity is given by B bits/sec, and duration of a GOP is T seconds, target bit rate of a GOP will be

$$R_{Target} = BT \quad \text{bits.} \tag{5.2}$$

Then the rate-control problem is given as follows:

$$\begin{cases} minimize \quad D = \sum_{i=1}^{N} D_i \\ according\ to \quad R = \sum_{i=1}^{N} R_i = R_{Target} \end{cases} \tag{5.3}$$

where D_i is the distortion, and R_i is the corresponding bit rate, the coding control parameter, of the frame f_i.

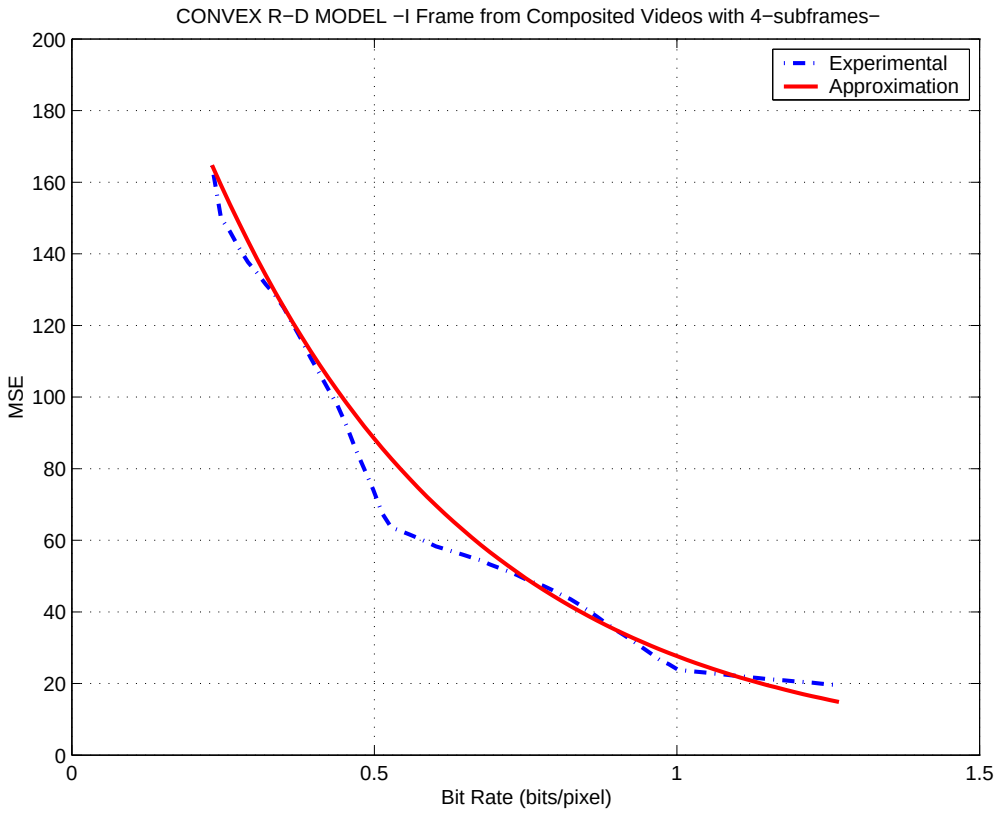


Figure 51: Convex R-D model

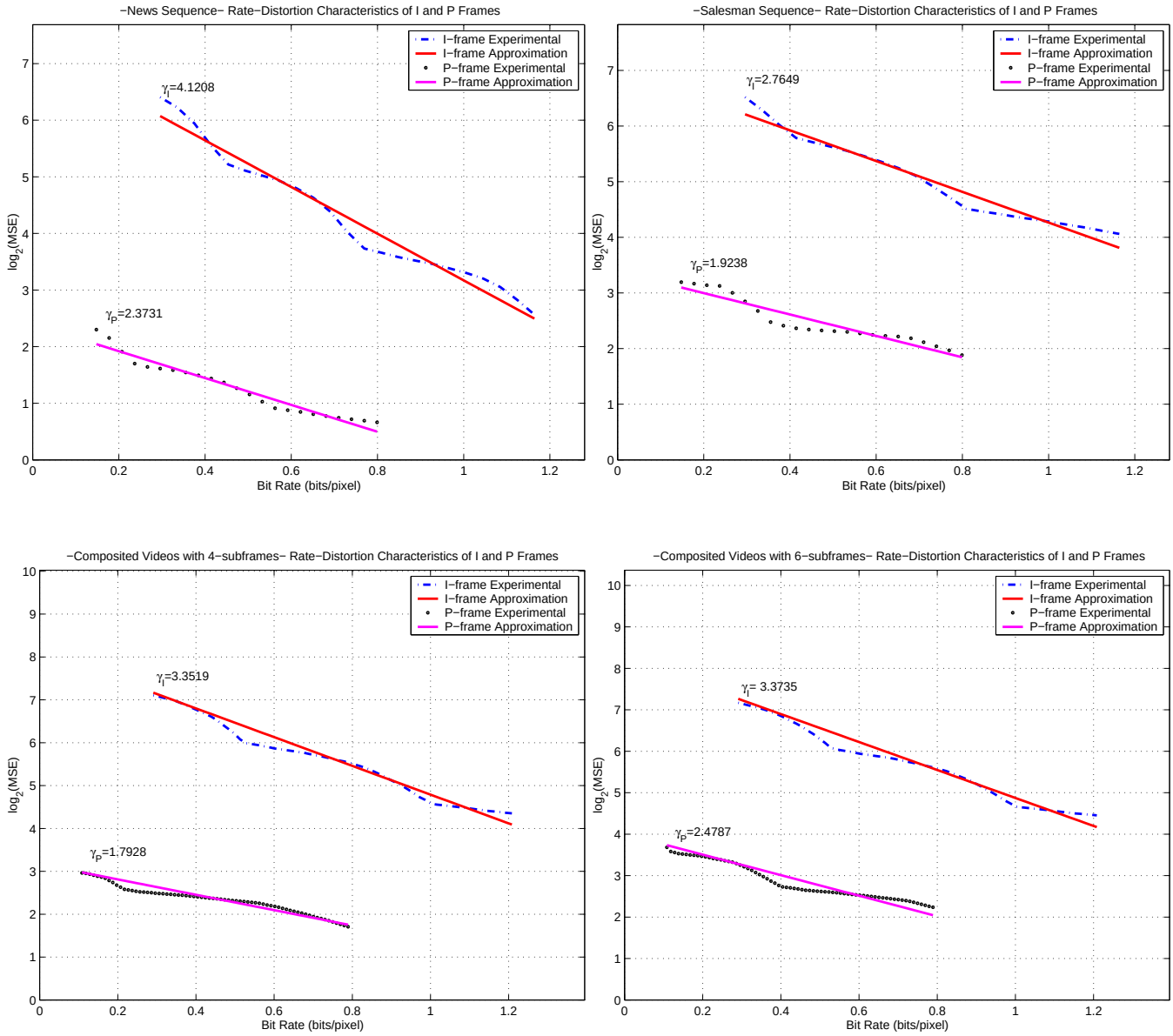


Figure 52: Rate-Distortion characteristics of first I and P frames from different video sequences

5.2 FRAME DEPENDENCY PROBLEM

In order to solve the bit rate constraint optimization problem in Eq. 5.3, one first performs the experiments to find the R-D characteristics of each frame given in Eq. 5.1, and then solves it via Lagrangian optimization. However, R-D curve of each frame is dependent on the R-D curve of previously coded frames. In other words, the prediction error corresponding the current frame depends on how previous frame has been encoded. Actually, for each (R_i, D_i) point in currently encoded frame, there is a different R-D curve for the next frame [44, 45]. This is called frame dependency problem.

Typically allocating more bits to an I frame improves the quality of motion compensation resulting in reducing the bit rates for the following P frames [42]. However bit rate distribution should be optimized to have the best average PSNR of a GOP.

Now first consider the error of motion compensation with respect to original frame which is given as

$$e(i,j) = c(i,j) - r(m[i,j]) \tag{5.4}$$

where $r(m[i,j])$ is motion compensated reference frame, and $c(i,j)$ is its predictively coded frame. Here $m[i,j]$ is the motion compensation vectors. Then the variance of the motion compensated residue is given as

$$\sigma_r^2 = E[e(i,j)^2] = E[\{c(i,j) - r(m[i,j])\}^2]. \tag{5.5}$$

Since at the decoder we only have encoded reference frame $\hat{r}(i,j)$ the actual residual variance will be

$$\hat{\sigma}_r^2 = E[\hat{e}(i,j)^2] = E[\{c(i,j) - \hat{r}(m[i,j])\}^2] \tag{5.6}$$

where $\hat{e}(i,j)$ is the actual residual error. This error can also be written as the summation of the residue of motion compensation with respect to the original reference frame and the error of the motion compensated reference frame as follows:

$$\hat{e}(i,j) = \{c(i,j) - r(m[i,j])\} + \{r(m[i,j]) - \hat{r}(m[i,j])\}. \tag{5.7}$$

In the same manner, we can also rewrite the variance of the motion compensated residue as

$$\hat{\sigma}_r^2 = \sigma_r^2 + E[\{r(m[i,j]) - \hat{r}(m[i,j])\}^2] \tag{5.8}$$

where the second component is the mean square error of the motion compensated reference frame.

There is a linear relationship between the mean square error of the motion compensated reference frame and that of the original reference frame as

$$E[r(m[i,j]) - \hat{r}(m[i,j])^2] \cong \alpha E[\{r(i,j) - \hat{r}(i,j)\}^2] \tag{5.9}$$

where α is frame dependency parameter [42]. Finally we can rewrite Eq. 5.8 as

$$\hat{\sigma}_r^2 = \sigma_r^2 + \alpha D \tag{5.10}$$

where D stands for coding distortion, which is the mean square error of the original reference frame given in Eq. 5.9. The linear relationship between the variance of the actual residue error and the mean square error of the original reference frame in Eq. 5.10, was also verified by the experiments as shown in Fig. 53.

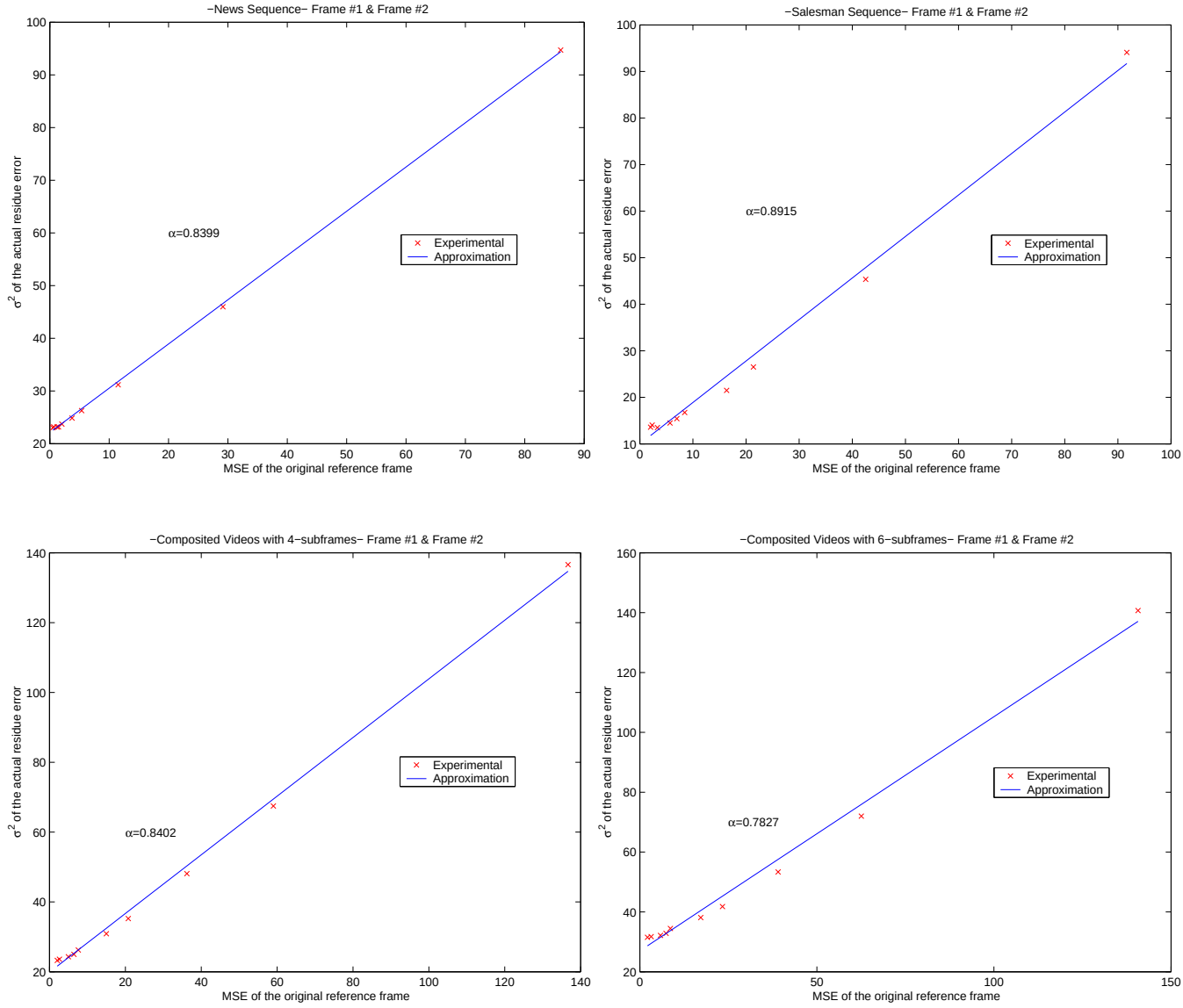


Figure 53: Relationship between the variance of the actual residue error and the mean square error of the original reference frame

5.3 USING LAGRANGIAN OPTIMIZATION TO ACHIEVE OPTIMIZED BIT RATE

Considering that the R-D functions of each frame are convex, the optimization problem given in Eq. 5.3 can be solved by using Lagrangian optimization. Now the problem can be rewritten as

$$minimize \;\; J(R_1, ..., R_N) = \sum_{i=1}^{N} D_i + \lambda \left(\sum_{i=1}^{N} R_i - R_{Target} \right) \tag{5.11}$$

where J is called Lagrangian or R-D cost, and λ is Lagrangian multiplier. Lagrangian multiplier λ is the absolute value of the slope of the tangency point of the R-D curve at where minimum distortion is achieved at given target bit rate. In Eq. 5.11, if λ is fixed, the rates that minimize this equation can be found. Now the constraint optimization in Eq. 5.3 becomes an unconstraint optimization problem which is easier to solve [42, 43, 44, 45]. From R-D models, having the rates $\{R_i(\lambda)\}_{i=1}^{N}$, Eq. 5.11 can be solved by searching a λ_0 such that

$$\sum_{i=1}^{N} R_i(\lambda_0) = R_{Target}. \tag{5.12}$$

Clearly from here, our aim is to find these optimized bit rates $\{R_i(\lambda)\}_{i=1}^{N}$ allowing minimum average distortion of a GOP.

Let us consider the partial derivatives of the Lagrangian cost with respect to bit rates R_i, which are zero at the optimum points, such as

$$\frac{\partial J}{\partial R_i} = 0, \quad i = 1, \; 2, \; ..., \; N \tag{5.13}$$

Also for each individual frame, R-D equation from Eq. 5.1 can be written as

$$D_i = \hat{\sigma}_i^2 2^{-\gamma_i R_i} \tag{5.14}$$

where the actual residual variance of frame f_i, $\hat{\sigma}_i$, is

$$\hat{\sigma}_i^2 = \sigma_i^2 + \alpha_i D_{i-1}. \tag{5.15}$$

Now consider the partial derivative in Eq. 5.13 for the last frame f_N in GOP,

$$\frac{\partial J}{\partial R_N} = \frac{\partial D_N}{\partial R_N} + \lambda = 0 \tag{5.16}$$

therefore

$$\frac{\partial D_N}{\partial R_N} = -\lambda = -\sigma_N^2 2^{-\gamma_N R_N} ln(2) \tag{5.17}$$

and since the last frame f_N is not a reference frame for other frames one can easily obtain the distortion of the frame f_N as

$$D_N = \frac{\lambda}{\gamma_N ln(2)}. \tag{5.18}$$

Similarly to Eq. 5.16, partial derivative of Lagrangian cost with respect to R_{N-1} can be found as

$$\frac{\partial J}{\partial R_{N-1}} = \frac{\partial D_{N-1}}{\partial R_{N-1}} + \frac{\partial D_N}{\partial R_{N-1}} + \lambda = 0 \tag{5.19}$$

and by using the Eq. 5.14 and Eq. 5.15, we get

$$\frac{\partial D_{N-1}}{\partial R_{N-1}}(1 + \alpha_N 2^{-\gamma_N R_N}) = -\lambda. \tag{5.20}$$

and therefore distortion of the frame f_{N-1} will be

$$D_{N-1} = \frac{\lambda}{\gamma_{N-1} ln(2)(1 + \alpha_N 2^{-\gamma_N R_N})}. \tag{5.21}$$

Since

$$\begin{aligned} 2^{-\gamma_N R_N} &= D_N/\sigma_N^2 \\ &= \frac{\lambda}{\gamma_N ln(2)\sigma_N^2} \end{aligned}$$

Eq. 5.21 becomes

$$D_{N-1} = \frac{\lambda}{\gamma_{N-1} ln(2)(1 + \alpha_N \frac{\lambda}{\gamma_N ln(2)\sigma_N^2})}. \tag{5.22}$$

If we consider the partial derivative of Lagrangian cost with respect to R_i, for $i \leq N-2$,

$$\frac{\partial J}{\partial R_i} = \frac{\partial D_i}{\partial R_i} + \frac{\partial D_{i+1}}{\partial R_i} + ... + \frac{\partial D_N}{\partial R_i} + \lambda = 0 \tag{5.23}$$

we will have that

$$\frac{\partial J}{\partial R_i} = \frac{\partial D_i}{\partial R_i} + \frac{\partial D_{i+1}}{\partial R_i} X_{i+1} + \lambda = 0 \tag{5.24}$$

where

$$X_{i+1} = \frac{\lambda}{\gamma_{i+1} ln(2) D_{i+1}}. \tag{5.25}$$

From Eq. 5.14 and Eq. 5.15,

$$D_{i+1} = \hat{\sigma}_{i+1}^2 2^{-\gamma_{i+1} R_{i+1}}$$

$$= (\sigma_{i+1}^2 + \alpha_{i+1} D_i) 2^{-\gamma_{i+1} R_{i+1}} \tag{5.26}$$

then Eq. 5.24 becomes

$$\frac{\partial J}{\partial R_i} = \frac{\partial D_i}{\partial R_i} X_i + \lambda = 0 \tag{5.27}$$

where

$$X_i = 1 + \alpha_{i+1} 2^{-\gamma_{i+1} R_{i+1}} X_{i+1}. \tag{5.28}$$

If we put Eq. 5.14 and Eq. 5.25 into Eq. 5.28 we will have

$$X_i = 1 + \frac{\alpha_{i+1} \lambda}{\hat{\sigma}_{i+1}^2 \gamma_{i+1} ln(2)} \tag{5.29}$$

and from Eq. 5.27, D_i will be

$$D_i = \frac{\lambda}{\gamma_i ln(2) X_i}. \tag{5.30}$$

Finally, by putting Eq. 5.15 and Eq. 5.29 into Eq. 5.30 a second order distortion function is obtained:

$$a_i D_i^2 + b_i D_i + c_i = 0, \quad i = 1, 2, ..., N-2 \tag{5.31}$$

where

$$a_i = \alpha_{i+1} \gamma_i \gamma_{i+1}$$

$$b_i = \sigma_{i+1}^2 \gamma_i \gamma_{i+1} + \alpha_{i+1} (\gamma_i - \gamma_{i+1}) \lambda / ln(2)$$

$$c_i = \sigma_{i+1}^2 \gamma_{i+1} \lambda / ln(2) \quad .$$

Solving Eq. 5.31, distortions will be

$$D_i = \frac{-b_i + \sqrt{b_i^2 - 4a_i c_i}}{2a_i}, \quad i = 1, 2, ..., N-2. \tag{5.32}$$

Now since we have the distortions, finally we can find the bit rates $\{R_i\}_{i=1}^{N}$ for each frame in a GOP by using the Eq. 5.14 and Eq. 5.15 as follows:

$$R_i = \begin{cases} \frac{1}{\gamma_1} log_2 \frac{\sigma_1^2}{D_1}, & i = 1 \\ \frac{1}{\gamma_i} log_2 \frac{\alpha_i D_{i-1} + \sigma_i^2}{D_i}, & i = 2, ..., N \end{cases} \tag{5.33}$$

To obtain the bit rates, one first needs to find the distortions given by Eq. 5.18, Eq. 5.22 and Eq. 5.32. Also Lagrangian multiplier λ is needed to be found. There are several simple algorithms to find λ, which one of them is bisection iteration method whose details can be found in [42].

This rate control scheme with the explained solution to the frame dependency problem has been shown to be very efficient for wavelet zerotree coders [42]. In the next section, we will compare this method with a piecewise linear R-D model scheme to show its effectiveness when used with the proposed DCT-based embedded zerotree coder.

5.4 COMPARISON OF THE CONVEX R-D MODEL WITH PIECEWISE LINEAR R-D MODEL

In [44], Silva et al. investigates rate control problem by using piecewise linear R-D model for embedded wavelet zerotree coding. In this section, we will compare this method by using it with DCT-based embedded zerotree coding against the method we use which has convex R-D model.

To solve the Lagrangian rate control optimization problem given in Eq. 5.11 by using the piecewise linear model, first R-D characteristics of each frame are estimated. Each piece of linear curve as shown in Fig. 54 is obtained by considering the beginning and the end point of each linear curve lies between the boundaries of consecutive dominant and subordinate passes [44]. Therefore to estimate the R-D characteristics of a frame, decoder decodes the

encoded frame for the rates corresponding to the breakpoints. Then following algorithm is used to find the optimum bit rate for given GOP:

1. *For each frame find the tangency point* $(R_i(\lambda), D_i(\lambda))$ *for given* λ,
2. *Compute the total bit rate* $R(\lambda)$,
3. *If the total bit rate,* $R(\lambda)$, *is not equal to the target bit rate,* R_{target}, *vary* λ *and go to Step 1, else the optimal bit rates are given by*$\{R_i(\lambda)\}_{i=1}^{N}$, *and stop.*

The values of λ here are found by determining the negatives of the set of slopes of all the linear pieces of the R-D curves of every frame [44]. In [44], Silva et al. propose an iterative method that copes with the frame dependency problem. In their method, they apply the rate control strategy described above and have the reconstructed frames for iteration n. Then rate allocation for the iteration $n+1$ is computed and so the reconstructed frames for iteration $n+1$ is obtained. This process is continued until the change in the distortion is below a threshold. However since this method requires several times encoding and decoding the frames of a GOP, we use the frame dependency parameter explained in the previous section.

The comparison results between the rate control method that uses convex R-D model and the one that uses piecewise linear R-D model are given in Fig. 55 and 56. In Fig. 55, PSNR results of the reconstructed composited video frames with four subframes are compared. In Fig. 56 PSNR values for the compositing case with six subframes are displayed. As shown in Table 13 and Table 14, average PSNR values obtained from convex R-D models are slightly better than the ones from piecewise R-D models. Therefore convex R-D model with the solution to the frame dependency problem generally achieves a better PSNR performance.

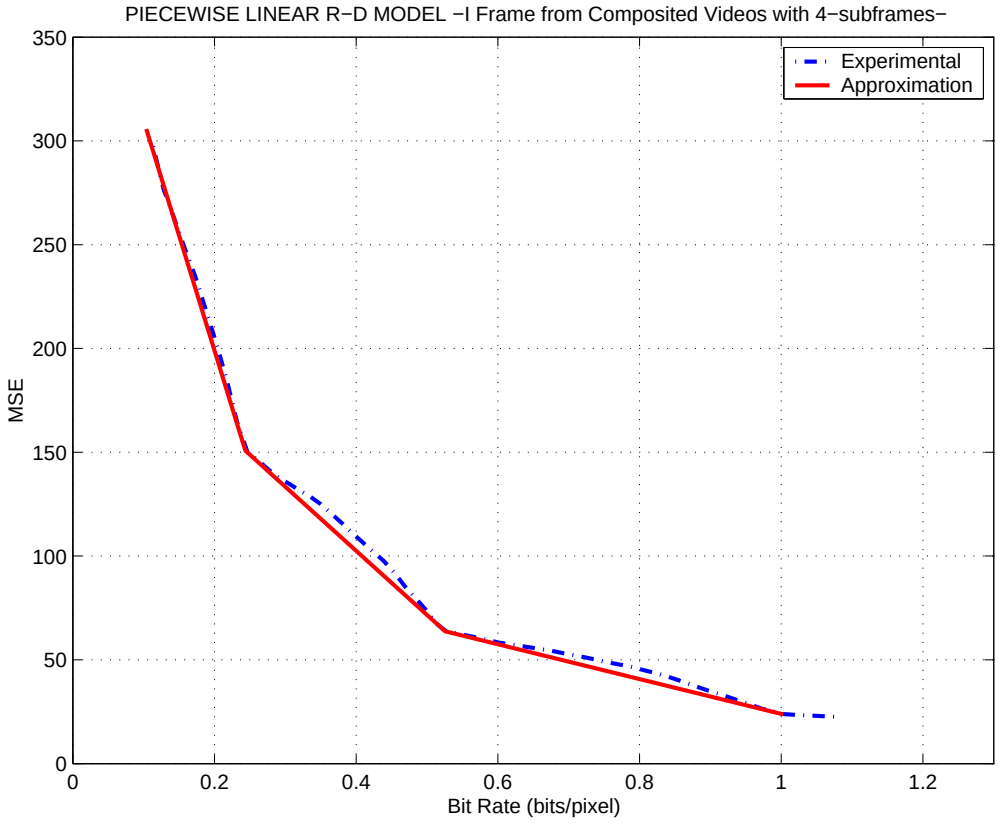


Figure 54: Piecewise linear R-D model

Table 13: Average PSNR comparisons of rate control with convex R-D model with piecewise linear R-D model for composited videos with four subframes

Bit Rate (bits/pixel)	Convex R-D Model	Piecewise Linear R-D Model
0.25	35.7915 dB	35.5865 dB
0.50	37.9902 dB	37.8615 dB
0.75	38.7083 dB	38.5828 dB
1.00	39.4099 dB	39.0432 dB

Table 14: Average PSNR comparisons of rate control with convex R-D model with piecewise linear R-D model for composited videos with six subframes

Bit Rate (bits/pixel)	Convex R-D Model	Piecewise Linear R-D Model
0.25	33.1050 dB	32.9113 dB
0.50	36.2953 dB	36.0333 dB
0.75	39.1545 dB	39.2050 dB
1.00	40.3763 dB	40.3048 dB

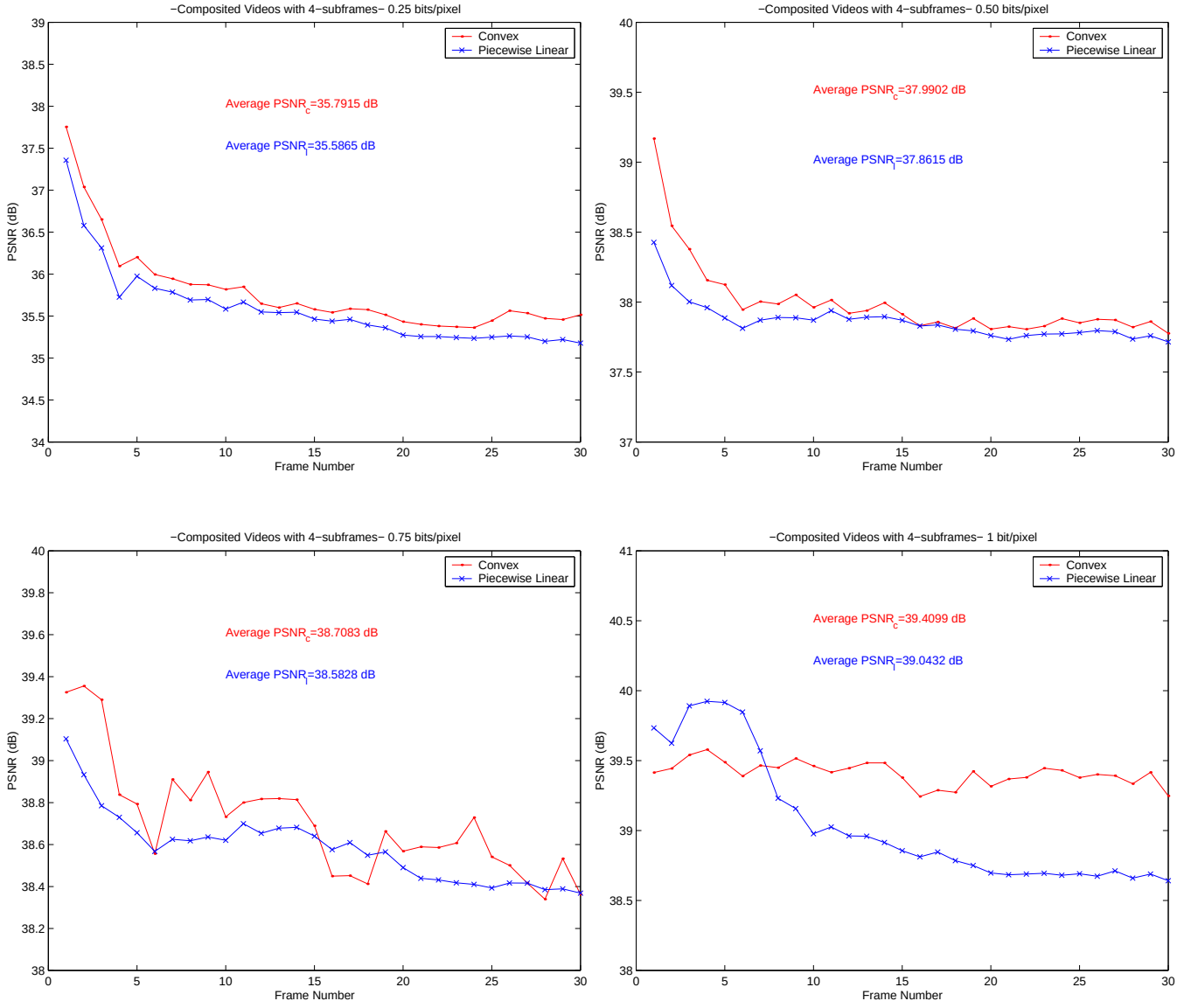


Figure 55: Comparison of R-D performances of the proposed convex model with piecewise linear model for composited videos with 4-subframes

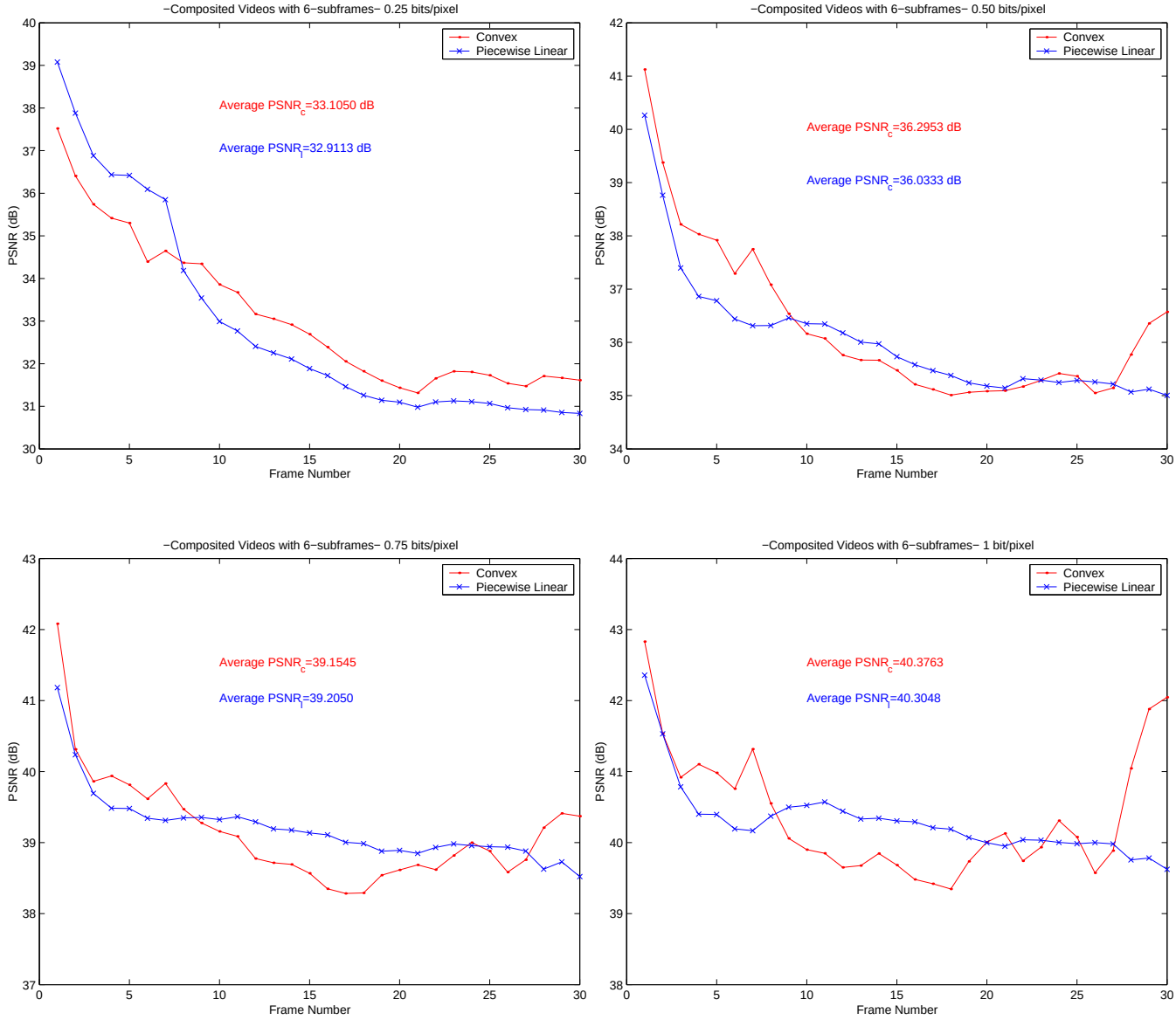


Figure 56: Comparison of R-D performances of the proposed convex model with piecewise linear model for composited videos with 6-subframes

5.4.1 Bit Rate Allocation at Subframe Layer

In this subsection, we investigate if subframe layer bit rate allocation is necessary for the proposed DCT-based embedded zerotree coding in the DCT compositing system. For this propose; after allocating the bit rates $\{R_i\}_{i=1}^{N}$ to each frame in a GOP, we divide the number of bits to the each subframe according to variance of each one such as,

$$R_{i,j} = w_{i,j} R_i \tag{5.34}$$

where $R_{i,j}$ is the allocated bit rate to subframe $f_{i,j}$ of frame f_i, and $w_{i,j}$ is the weight of the bit rate of subframe $f_{i,j}$ obtained from the variances of each subframe as

$$w_{i,j} = \frac{\sigma_{i,j}}{\sum_{k=1}^{K} \sigma_{i,k}} \tag{5.35}$$

where $\sigma_{i,j}$ is the variance of subframe $f_{i,j}$ of frame f_i consisting of K subframes. By using this approach we distribute the bit rates among the subframes according to the activities in each one. The average PSNR results are shown in Table 15 and Table 16 for composited videos with four subframes and with six subframes, respectively. As seen from the tables subframe layer bit rate allocation does not have any advantages on improving the quality of composited videos. The reason is that since embedded zerotree coding uses successive approximation quantization, DCT coefficients are encoded by significance importance eliminating the evaluation of subframe layer bit rate allocation.

Table 15: Average PSNR comparisons of rate control with and without subframe layer bit rate allocation for composited videos with four subframes

Bit Rate (bits/pixel)	Rate Cont. w./ Sub. Bit Allo.	Rate Cont. w.o./ Sub. Bit Allo.
0.25	35.0012 dB	35.7915 dB
0.50	37.7122 dB	37.9902 dB
0.75	38.5595 dB	38.7083 dB
1.00	39.0206 dB	39.4099 dB

Table 16: Average PSNR comparisons of rate control with and without subframe layer bit rate allocation for composited videos with six subframes

Bit Rate (bits/pixel)	Rate Cont. w./ Sub. Bit Allo.	Rate Cont. w.o./ Sub. Bit Allo.
0.25	32.9707 dB	33.1050 dB
0.50	36.2666 dB	36.2953 dB
0.75	37.6953 dB	39.1545 dB
1.00	39.7483 dB	40.3763 dB

6.0 CONCLUSIONS AND FUTURE WORK

In this work we implement a DCT domain video compositing system with DCT-based embedded zerotree coding for multi-point video conferencing. Operating the compositing process fully in the DCT domain decreases the overall computational complexity and improves the composited video quality. We use DCT domain transcoders to decode different incoming video sequences. Motion compensation in the DCT domain is processed faster than the one in the spatial domain, because inverse DCT transform is not used. Also sparse matrices, which are used for windowing and shifting of the DCT blocks for motion compensation propose, decrease decoding time in the transcoding process.

For resizing, we come up with a new DCT decimation/interpolation method, which can be used with different decimation factors including rational numbers beside integers according to the number of incoming video streams to be composited. Decimation with rational decimation factor is applied with an increment of number of computations. We compare our PSNR results for the decimation factor of $N = 2$ with those of [12] of Dugad et al. since they only have the results for this decimation factor. The computational complexity of the other decimation factors are also shown. For the decimation by $N = 2$, our algorithm has the least computational complexity with the one in [12]. Furthermore we obtain slightly better PSNRs than those in work of Dugad et al. With a small increase in the number of computations we get higher PSNRs. In this case, our algorithm is still better than other algorithms such as [5] and spatial domain algorithm in terms of computational complexity.

The composited videos are encoded efficiently by using the DCT-based embedded zerotree coder which was originally developed for wavelet coding [15], and also used with DCT-based image and video coding [18, 20]. To use the zerotree coder with DCT coefficients they are rearranged into a hierarchical structure similar to the wavelet subbands.

Adaptive arithmetic coding is used to encode the symbols obtained from dominant and subordinate passes considering the usefulness of the arithmetic coding when dealing with sources with small alphabets. We also use the advantage of the embedded bit stream property of the zerotree coding. As each symbol is encoded by adaptive arithmetic encoder, the number of the bits at the output is counted, so when the desired bit budget is reached the coding is terminated. We obtain better results by using the DCT embedded zerotree coder than conventional DCT encoder that uses regular scalar quantizer. The improvement of the composited videos are 1-2.7 dB on average.

We also introduce the proposed DCT decimation/interpolation and zerotree coding to encode single video streams. We use integer and rational decimation factors with the proposed coding. For intraframe case, we show that coding with the highest decimation factor obtains better results than the others at very low bit rates since it has the smallest decimated video frame. However as the number of bits increases the efficiency of the high decimation factor decreases. This is because decimated frames requires less bits than the full frames to be encoded. So after a certain number of bits, decimated video frame does not require more bits and any additional bits will not increase the quality of the decimated video resulting saturation in PSNR of the reconstructed video. For both intraframe and interframe cases, our method gives better PSNRs than the regular coding from 7.47 kilobits/second up to 45 kilobits/second for the case where the decimation factor $N = 2$. Therefore the proposed single stream video coding with the introduced DCT decimation/interpolation is very efficient at low bit rates. Also other decimation factors can be applied to to both intraframe and interframe coding cases.

We also use another zerotree coding method implemented for embedded wavelet coders in [21] to our compositing system in order to increase the efficiency. Beside the efficiency of the encoder, since data symbols obtained at the output of the encoder is completely binary, the performance of adaptive arithmetic encoder increases. Consequently the quality of the reconstructed frames are improved.

To decrease the blocking effects at low bit rates, we implement LCT [37] based embedded zerotree coding. Since LCT already includes DCT, we easily adopt this method into our system with an increase in computational complexity. However, since we consider that the

incoming video streams are in the DCT domain, we apply inverse DCT to obtain frames in the spatial domain. The reason for this is that the folding operation is only applied in the spatial domain. This increases the computational complexity. Nevertheless if the incoming frames are in the LCT domain, the inverse DCT operation is excluded. Consequently computational load decreases. The other advantage of the LCT in the proposed system is that the windowing and shifting matrices of the transcoders are compatible with LCT blocks of video frames. Therefore there is no need to derive new windowing and shifting matrices for LCT case since motion compensation in the LCT domain can be realized with these matrices. Thus DCT domain compositing system if fully compatible with video sources using DCT or LCT domain coding.

Finally, we use a convex R-D model in bit rate control [42]. To distribute the bits to each frame optimally, Lagrangian optimization is used. Frame dependency problem is solved by computing the frame dependency parameter which is obtained form the linear relationship between the variance of the actual residue error and the distortion of the original reference frame [42]. Since embedded zerotree coder does not require evaluation of quantization parameter, which is the case in regular quantization, we only need to solve the bit rate problem in the frame layer. Also unlike regular quantization the coding control parameter is the allocated bit rate to each frame in a GOP. Therefore we get exact target bit rate. However regular quantization requires a feedback to reevaluate the quantization parameters to reach the target bit rate. Still it does not guarantee to achieve the target bit rate precisely requiring usage of a buffer. We also show that there is no need to obtain the statistics of each subframe to distribute the allocated bit rate of a frame to each subframe. The reason for this is that the embedded zerotree coding uses successive approximation quantization that allows the most significant DCT coefficients to be encoded first whether they are in any subframe. In the same manner, the other DCT coefficients are encoded in the descending significance order. We also compare the performances of the bit rate control methods using convex [42] and piecewise linear models [44]. Bit rate control with convex model achieves slightly higher PSNRs than that with piecewise linear model.

As future works, first it is possible to improve the coding efficiency of the adaptive arithmetic encoder by using different contexts, such as group of neighbor symbols [24, 15, 21].

Since real time video compositing requires fast operations, the approach in [16] can be used to identify zerotrees faster than the approach in [15]. There are also other zerotree coding methods such as 3-D coefficient tree structure for 3-D wavelet in [17] that can be applied to DCT zerotree coding to achieve more compression or quality. It is also worth to compare the overall runtime of the spatial domain compositing system with that of the DCT domain compositing system using embedded zerotree coding for both methods, DCT-EZT and DCT-SPIHT. Also processing times of regular quantization and embedded zerotree coding can be compared to investigate the complexity of the zerotree coding over the regular one. It must be stated that a slight increase in PSNR may not be visible to the viewers. Therefore, for a faster compositing, our DCT decimation/interpolation method can be used with smaller values of q than those used in this work. In the same manner, after comparing the processing time of each zerotree coding methods, faster one can be chosen even if it achieves lower PSNR values than the other one. To make the overall process faster, computational complexity to obtain and to process the two R-D models can be compared, and faster one can be chosen. Another future work that can be implemented is to use the obtained R-D models of the frames of the first GOP to allocate the target bit rate among the frames of the next GOPs instead of obtaining new R-D models for each GOP. Although this may decrease average PSNR of a GOP, it may be useful for real-time compositing to decrease the coding delay. Another reason leading to this conclusion is that in video conferencing, the background information is stable, and the person speaking generally does not move significantly. Therefore, statistics of consecutive GOPs may not differ considerably. Other bit rate control schemes [46, 47] can also be considered to be compared with the performance of the bit rate control schemes used in this work. Also a research on a possible folding operation in the DCT domain can be considered if it decreases the computational complexity of the LCT-based compositing system.

APPENDIX A

SPARSE MATRICES IN FAST DCT TRANSCODER

The sparse matrices, which their products give the 8×8 DCT matrix S_8 in Equation 2.14, is given as follows. The diagonal matrix D is

$$D = \begin{bmatrix} 0.3536 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.2549 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.2706 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.3007 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.3536 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.4500 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.6533 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1.2814 \end{bmatrix}.$$

The second one is a permutation matrix which is defined as

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}.$$

The other sparse matrices, B_1, B_2, M_1, A_1, A_2, and A_3 are given as follows.

$$B_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 \end{bmatrix}$$

$$B_2 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 \end{bmatrix}$$

$$M_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.7071 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -0.9239 & 0 & -0.3827 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.7071 & 0 & 0 \\ 0 & 0 & 0 & 0 & -0.3827 & 0 & 0.9239 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_1 = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_2 = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_3 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix}.$$

APPENDIX B

ADAPTIVE ARITHMETIC CODING

Arithmetic coding effectively utilizes the redundancies present in zerotree and subordinate arrays for lossless compression producing embedded bitstreams. The detailed algorithm of the adaptive arithmetic coding is given in [24]. Although arithmetic coding is more complex than Huffman coding, it is especially useful when processing the sources with small alphabets. Another advantage of the arithmetic coding is that a system with multiple arithmetic codes is very easy to implement [28]. Additionally, it is very easy to adapt arithmetic codes to changing symbol statistics. This is done by estimating the probabilities of the input alphabet that is simply keeping a count of the symbols as they are coded. There is no requirement of preserving a tree as with adaptive Huffman codes, and also there is no need to generate a code a priori as in the Huffman coding. To reduce the complexity it is also possible to develop multiplication-free arithmetic coders [28]. The flowchart of the algorithm is illustrated in Fig. 57.

For adaptivity, a histogram count of the symbols, which is the model of adaptive arithmetic coder, is used. Different from the adaptive arithmetic coder, fixed model arithmetic coders use predetermined table of frequency counts of the symbols. In practice, adaptive model arithmetic coders outperform the fixed model ones in terms of compression ratio [24, 25, 27, 30, 31]. Thus in the experiments, adaptive model was chosen. The distribution of the probabilities for the symbols are calculated as each symbol is encoded. Initially, all counts of the symbols are the same, and are set to one. For a better adaptation, the initial counts may be set to predetermined frequencies presenting the overall statistics of the sym-

bols better. As each symbol is seen, the counts are updated. Both encoder and decoder use the same initial counts and the same algorithm for updating the model to be synchronized. Therefore, there is no need to transmit additional information to the decoder for updating. When sum of the counts of all symbols reaches a maximum cumulative count, each symbol frequency count is divided by two to control the learning rate for adaptation. This gives more weighting to the recent symbols than the earlier ones as stated in [24]. This adaptation method with a limited past histogram reduces the bit rate more than %30 below the first order entropy of the symbols [27]. If some symbols, which have been seen rarely in the past, occur more frequently lately hence they require lower bit rates [24, 27, 29, 32].

Maximum cumulative counts of the symbols are predetermined. In the experiments, 256 maximum cumulative count was used for both zerotree and subordinate symbols. The other counts like 64, 512, and 1024 were also used to see if better compression results were achieved. Then 256 was chosen since it was found to be the most suitable maximum histogram count.

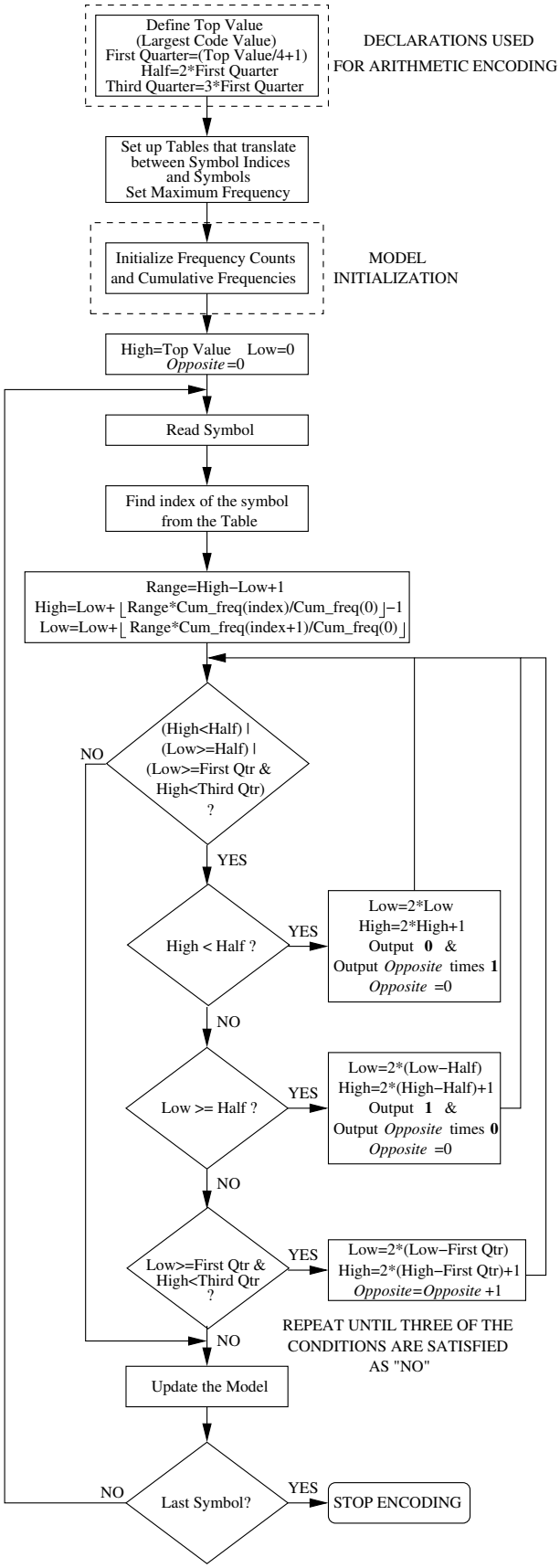


Figure 57: Flowchart of the adaptive arithmetic encoder

BIBLIOGRAPHY

[1] L. F. Chaparro, and C. C. Li, "New Algorithms for DCT-based transcoding and compositing in multi-point video conferencing," Final Report of PDG Project, Univ. of Pittsburgh, Apr. 2003.

[2] S. A. Martucci, "Image resizing in the discrete cosine transform domain," in *Proc. IEEE International Conf. Image Proc.*, Oct. 1995, Vol. 2, pp. 244-247.

[3] S.-F. Chang, and D. G. Messerschmitt, "Compositing motion-compensated video within the network," *4th IEEE ComSoc Intl. Workshop on Multimedia Communications*, pp. 40-56, Monterey, CA, Apr. 1992.

[4] S.-F. Chang, and D. G. Messerschmitt, "A new approach to decoding and compositing motion-compensated DCT-based images," *Proc. IEEE Intl. Conf. Acoustic, Speech, and Signal Processing*, Vol. 5, pp. 421-424, Minneapolis, MN, Apr. 1993.

[5] S.-F. Chang, and D. G. Messerschmitt, "Manipulation and compositing of MC-DCT compressed video," *IEEE Journal on Selected Areas in Commun.*, Vol. 13, No. 1, pp. 1-11, Jan. 1995.

[6] Y. Noguchi, D. G. Messerschmitt, and S.-F. Chang, "MPEG video compositing in the compressed domain," *Proc. IEEE Intl. Symp. Circuits and Systems*, Vol. 2, pp. 596-599, May 1996.

[7] M. Song, A. Cai, J.-a. Sun, "Motion estimation in DCT domain," *Proc. IEEE Comm. Technology, ICCT'96*, Vol. 2, pp. 670-674, May 1996.

[8] N. Merhav, and V. Bhaskara, "A transform domain approach to spatial domain image scaling," *IEEE Intl. Conf. Acoustic, Speech, and Signal Processing*, Vol. 4, pp. 2403-2406, Atlanta, GA, May 1996.

[9] N. Merhav, and V. Bhaskaran, "A fast algorithm for DCT-domain inverse motion compensation," *Proc. IEEE International Conf. Acoustics, Speech, and Signal Processing*, Vol. 4, pp. 2307-2310, May 1996.

[10] N. Merhav, and V. Bhaskaran, "Fast algorithms for DCT-domain image downsampling and for inverse motion compensation," *IEEE Trans. Circuits and Systems for Video Technology*, Vol. 7, No. 3, pp. 468-476, June 1997.

[11] J. Song, and B.-L. Yeo, "A fast DCT domain inverse motion compensation algorithm based on shared information in a macroblock," *Proc. Asimolar Conference on Signals, Systems and Computers*, Vol. 1, pp. 845-849, Nov. 1998.

[12] R. Dugad, and N. Ahuja, "A fast scheme for image size change in the compressed domain," *IEEE Trans. Circuits and Syst. for Video Technology*, pp. 461-474, Apr. 2001.

[13] J. Jian, G. Feng, "The spatial relationship of DCT coefficients between a block and its sub-blocks," *IEEE Trans. Signal Proc.*, pp. 1160-1169, May 2002.

[14] J. Mukherjee, and S. K. Mitra "Image resizing in the compressed domain using subband DCT," *IEEE Trans. Circuits and Syst. for Video Tech.*, pp. 620-627, Jul. 2002.

[15] J. M. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," *IEEE Trans. Signal Proc.*, Vol. 41, No. 12, pp. 3445-3462, Dec. 1993.

[16] J. M. Shapiro, "A fast technique for identifying zerotrees in the EZW algorithm," *IEEE Intern. Conf. on Acoustics, Speech, and Signal Proc., ICASSP-96*, Vol. 3, pp. 1455-1458, May 96.

[17] C. He, J. Dong, Y. F. Zheng, Z. Gao, "Optimal 3-D coefficient tree structure for 3-D wavelet video coding," *IEEE Trans. Circuits and Syst. for Video Tech.*, Vol. 13, pp. 961-972, Oct. 2003.

[18] Z. Xiong, O. G. Guleryuz, and M. T. Orchard, "A DCT-based embedded image coder," *IEEE Trans. Signal Proc.*, Vol. 3, No. 11, pp. 289-290, Nov. 1996.

[19] D. M. Monro, and G. J. Dickson, "Zerotree Coding of DCT coefficients," *IEEE Intern. Conf. Image Proc.*, Vol. 2, pp. 625-628, Oct. 1997.

[20] Y.-A. Jeong, and C.-K. Cheong, "A DCT-based embedded image coder using wavelet structure of DCT for very low bit rate video codec," *IEEE Trans. Cons. Elec*, Vol. 44, No. 3, pp. 500-507, Aug. 1998.

[21] A. Said, and W. A. Pearlman, "A new, fast, and efficient image codec based on set partitioning in hierarchical trees," *IEEE Trans. Circuits and Syst. for Video Technology*, Vol. 6, No. 3, pp. 243-250, June 1996.

[22] E. Khan and M. Ghanbari, "Video Coding with Virtual Set Partitioning in hierarchical tree," *IEEE Inter. Symposium on Circuits and Systems*, Vol. 1, pp. 449-452, May 2002.

[23] A. P. Azcarraga, M. R. Lim, "2-D order of self-organizing kristal maps," *IJCNN Intern. Joint Conf. Neural Networks*, Vol. 1, pp. 510-513, July 1999.

[24] I. H. Witten, R. M. Neal, and J. G. Cleary, "Arithmetic coding for data compression," *Commun. ACM*, Vol. 30, No. 6, pp. 520-540, June 1987.

[25] E. Baum, V. Harr, and J. Speidel, "Improvement of H.263 encoding by adaptive arithmetic coding," *IEEE Trans. Circuits and Syst. for Video Technology*, Vol. 10, No. 5, Aug. 2000.

[26] *ITU-T Recommendation H.263*, Video coding for low bit rate communication, Feb. 1998.

[27] M. Ghanbari, *Video Coding, An Introduction to Standart Codecs.* The Institution of Electirical Engineers, London, UK, 1999.

[28] K. Sayood, *Introduction to Data Compression.* Morgan Kauffmann Publishers, Inc., San Francisco, CA, 1996.

[29] V. Bhaskaran, K. Konstantinides, *Image and Video Compression Standarts.* Kluwer Academic Publishers, Norwell, MA, 1997.

[30] M. Nelson, J.-L. Gially, *The Data Compression Book.* M&T Books, New York, NY, 1996.

[31] L. Wall, K. Ferens, and W. Kinsner, "Real-time dynamic arithmetic coding for low bit-rate channels," in *Proc. IEEE Conf. Commun., Computers, and Power in the Modern Env.*, Saskaaton, Canada, May 1993, pp. 381-391.

[32] P. G. Howard, J. S. Vitter, "Analysis of arithmetic coding for data compression," in *Proc. Data Compression Conf.*, Snowbird, UT, Apr. 1991, pp. 3-12.

[33] C.-K. Cheong, K.-S. Cho, and S.-W. Lee, "Significance tree image sequence coding with DCT-based pyramid structure," in *Proc. IEEE International Conf. Image Proc.*, Vancouver, Canada, Sep. 2000, Vol. 2, pp. 859-862.

[34] H. S. Malvar, "The LOT: a link between block transform coding and multirate filter banks," *IEEE Intern. Symp. Circuits and Systems*, pp. 835-838, Aug. 1988.

[35] H. S. Malvar, D. H. Staelin, "The LOT: transform coding without blocking effects," *IEEE Trans. Acoustics, Speech, and Signal Processing*, Vol. 37, No. 4, pp. 553-559, Apr. 1989.

[36] H. S. Malvar, *Signal processing with lapped transforms.* Artech House Publishers, Norwood, MA, 1992.

[37] G. Aharoni, A. Averbuch, R. Coifman, M. Israeli, "Local Cosine Transform - A Method for the Reduction of the Blocking Effect in JPEG," *Journal of Mathematical Imaging and Vision*, pp. 7-38, 1993.

[38] K. Ramchandran, A. Ortega, M. Vetterli, "Bit allocation for dependent quantization with applications to multiresolution and MPEG video coders," *IEEE Trans. Image Processing*, Vol. 3, No.5, pp. 533-545, Sept. 1994.

[39] L.-J. Lin, A. Ortega, and C.-C. J. Kuo, "Rate control using spline-interpolated R-D characteristics," in *Proc. VCIP*, Orlando, FL, pp. 111-122, Mar. 1996.

[40] T. Chiang, and Y.-Q. Zhang, "A new rate control scheme using quadratic rate distortion model," *IEEE Trans. Circuits and Systems for Video Technology*, Vol. 7, No. 1, pp. 246-250, Feb. 1997.

[41] J. Ribas-Corbera, and S. Lei, "Rate control in DCT video coding for low-delay communications," *IEEE Trans. Circuits and Systems for Video Technology*, Vol. 9, No. 1, pp. 172-185, Feb. 1999.

[42] P.-Y. Cheng, J. Li, and C.-C. J. Kuo, "Rate control for an embedded wavelet video coder," *IEEE Trans. Circuits and Systems for Video Technology*, Vol. 7, No. 4, pp. 696-702, Aug. 1997.

[43] Guido M. Schuster, Aggelos K. Katsaggelos *Rate-Distortion Based Video Compression.* Kluwer Academic Publishers, Boston / Dordrecht / London, 1997.

[44] E. A. B. da Silva, R. Caetano, "A rate control strategy for embedded wavelet video coders in an MPEG-4 framework," *Global Telecommunications Conference*, pp. 199-203, 1999.

[45] A. Ortega and K. Ramchandran, "Rate-distortion methods for image and video compression," *IEEE Signal Processing Magazine*, pp. 23-50, Nov. 1998.

[46] Z. He, and S. K. Mitra, "A linear source model and a unified rate control algorithm for DCT video coding," *IEEE Trans. Circuits and Systems for Video Technology*, Vol. 12, No. 11, pp. 970-982, Nov. 2002.

[47] Z. He, and S. K. Mitra, "Optimum bit allocation and accurate rate control for video coding via ρ-domain source modeling," *IEEE Trans. Circuits and Systems for Video Technology*, Vol. 12, No. 10, pp. 840-849, Oct. 2002.