Arbitrary Block-size Transform Video Coding

FONG, Chi Keung

A Thesis Submitted in Partial Fulfillment of
the Requirements for the Degree of

Doctor of Philosophy

in

Electronic Engineering

The Chinese University of Hong Kong

March 2011

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346

摘要

變換是一種很重要的視頻編解碼工具。它不但將圖像中像素的關聯性打破，有效地移除當中的冗贅，從而達至壓縮視頻資訊的效果。研究發現，大部份的碼流都是用在變換係數編碼。如果能採用更有效的變換，減少這部份的碼流，就能有效地改進現有的編碼效率。一直以來，視頻及圖像編碼一般都只採用第八階的變換。直至近年，最新的編解碼標準（例如H.264/AVC）開始採用兩種或以上的變換來壓縮視頻資訊，例如同時採用第四階及第八階變換來壓縮視頻資訊。此等採用多種不同大小變換的技術，稱爲可變塊大小變換或自適應塊大小變換。有研究指出更高階的變換，例如第十六階，可以更有效的提高在高清視頻序列上的編碼效果。因此，第十六階變換亦被加入到 ABT 系統當中。

三種不同而且全新的第十六階變換將會在本篇論文中提出，並詳細展示，進行各種的分析，以及對現有的第十六階變換作比對。這三種新的第十六階變換亦會在最新的解碼標準上實現，形成一個新的 ABT 系統，當中第四階、第八階以及第十六階的變換都並存在系統內，可以自由選取。而選取的方法是基於率失真優化，根據每種變換的編碼表現，每個宏塊會選用表現最好，編碼效果最好的變換。不同的宏塊可以有不同的變換選

擇。實驗證實所提出的三種換有非常良好的表現，各自有不同的表現，但都能有效減少所需碼流，提供更佳的資料壓縮，對畫像的主觀以及客觀的質量都有所增益。

除了用於編碼的高階變換外，本篇論文亦會牽涉一種利用變換而達致快速模式匹配的方法—快速沃爾什搜索。這種快速搜索方法不但快捷，而且準確。它可的準確度跟傳統的快速全搜索相約，但複雜度則大幅降低。它亦在 H.264/AVC 的平台上實現，用作運動預測的工具。實驗顯示它比傳統的快速運動估計方法更加優勝。不但作出準確估計，而且不需要煩瑣的運算，有效地改善現有的編碼系統。

最後，本篇論文亦會簡述一些對變換係數的統計研究。變換係數的統計對很多圖像及視頻處理方法都很重要，這些方法都是建基於一些係數統計上的數據與假設。若果可以對這些統計數據作出更深入準確的理解，就可以改善圖像及視頻處理方法，例如在編碼系統上的碼流控制等。傳統上，大都認爲係數是呈拉普拉斯分佈，包括預測殘餘的係數都有相同的假設。本篇論文的研究發現並不是每種預測殘餘都是呈拉普拉斯分佈。研究發現大部份的幀內預測殘餘都是呈柯西分佈，而幀間預測殘餘則是呈拉普拉斯分佈的。這一發現可以更有效改善現有的視頻處理方法。

Abstract

Transform is a very important coding tool in video coding. It decorrelates the pixel data and removes the redundancy among pixels so as to achieve compression. Traditionally, order-8 transform is used in video and image coding. Latest video coding standards, such as H.264/AVC, adopt both order-4 and order-8 transforms. The adaptive use of more than one transforms of different sizes is known as Arbitrary Block-size Transform (ABT). Transforms other than order-4 and order-8 can also be used in ABT. It is expected larger transform size such as order-16 will benefit more in video sequences with higher resolutions such as 720p and 1080p sequences. As a result, order-16 transform is introduced into ABT system.

In this thesis, the development of simple but efficient order-16 transforms will be shown. Analysis and comparison with existing order-16 transforms have been carried out. The proposed order-16 transforms were integrated to the existing coding standard reference software individually so as to achieve a new ABT system. In the proposed ABT system, order-4, order-8 and order-16 transforms coexist. The selection of the most appropriate transform is based on the rate-distortion performance of these transforms. A remarkable improvement in coding performance is shown in

the experiment results. A significant bit rate reduction can be achieved with our proposed ABT system with both subjective and objective qualities remain unchanged.

Three kinds of order-16 orthogonal DCT-like integer transforms are proposed in this thesis. The first one is the simple integer transform, which is expanded from existing order-8 ICT. The second one is the hybrid integer transform from the Dyadic Weighted Walsh Transform (DWWT). It is shown that it has a better performance than simple integer transform. The last one is a recursive transform. Order-2N transform can be derived from order-N one. It is very close to the DCT. This recursive transform can be implemented in two different ways and they are denoted as LLMICT and CSFICT. They have excellent coding performance. These proposed transforms are investigated and are implemented into the reference software of H.264 and AVS. They are also compared with other order-16 orthogonal integer transform. Experimental results show that the proposed transforms give excellent coding performance and ease to compute.

Besides ABT with higher order transform, a transform based template matching is also investigated. A fast method of template matching, called Fast Walsh Search, is developed. This search method has similar accuracy as exhaustive search but significantly lower computation requirement.

Prior knowledge of the coefficient distribution is a key to achieve better coding performance. This is very useful in many areas in coding such as rate control, rate distortion optimization, etc. It is also shown that coefficient distribution of predicted residue is closer to Cauchy distribution rather than traditionally expected Laplace distribution. This can effectively improve the existing processing techniques.

Publication List

Journal Papers

1. J. Dong, K.N. Ngan, C.K. Fong, W.K. Cham, "*2D Order-16 Integer Transforms for HD Video Coding*," IEEE Trans. on CASVT, vol.19, no.10, October 2009, pp.1463-1474.

2. C.M. Mak, C.K. Fong, W.K. Cham, "*Fast Motion Estimation for H.264/AVC in Walsh Hadamard Domain,*" IEEE Trans. on CASVT, vol.18, no.6, June 2008, pp.735-745.

Conference Papers

1. C.K. Fong, W.K. Cham, "*Simple Order-16 Integer Transform for Video Coding*," ICIP, Page(s): 161-164, 2010.

2. J. Dong, K.N. Ngan, C.K. Fong, W.K. Cham, "*A Universal Approach to Developing Fast Algorithm for Simplified Order-16 ICT*," IEEE International Symposium on Circuits and Systems, ISCAS, Page(s): 281 - 284 2007.

Standard Proposals

1. C.K. Fong, W.K. Cham, Y. Liu, K.M. Cheng, "*Adaptive Block-size Transform towards AVS 2.0,*" AVS Video Proposal AVS-M2666, Guangzhou, Dec. 2009.

2. C.K. Fong, W.K. Cham, K.N. Ngan, Y. Liu, K.M. Cheng, "*An investigation of Order-16 Transform in M2606 ABT*," AVS Informative Proposal AVS-M2657, Wuxi, Sept. 2009.

3. C.K. Fong, W.K. Cham, K.N. Ngan, Yu Liu, K.M. Cheng, "*Adaptive Block-size Transform towards AVS 2.0,*" AVS Video Proposal AVS-M2647, Wuxi, Sept. 2009.

4. W.K. Cham, C.K. Fong, Y.L. Fong, K.N. Ngan, Y. Liu, K.M.Cheng, "*Adaptive Block-size Transform towards AVS 2.0,*" AVS Video Proposal AVS-M2610, Wuxi, Sept. 2009.

5. X. Mao, Y. Wang, Y. He, W.K. Cham, C.K. Fong, J. Dong, K.N. Ngan, H. M. Wong, L. Wang, Y. Huo, T. Pun, C. Cheng, "*AVS 自适应块大小编码技术*," AVS Video Proposal AVS-M2372, Xiamen, June 2008.

6. W.K. Cham, C.K. Fong, J. Dong, K.N. Ngan, H.M. Wong, L. Wang, Y. Huo, T. Pun "*Adaptive Block-size Transform for AVS-X*," AVS Video Proposal AVS-M2284, Lijiang, March 2008.

7. W.K. Cham, C.K. Fong, J. Dong, K.N. Ngan, H.M. Wong, L. Wang, Y. Huo, T. Pun, "*Adaptive Block-size Transform for AVS-X and AVS-S profile*," AVS Video Proposal AVS-M2182, Shanghai, Dec. 2007.

Pending Patents

1. W.K. Cham, C.K. Fong, "*DEVICES AND METHODS FOR TRANSFORMING CODING COEFFICIENTS OF VIDEO SIGNALS*," US Non-Provisional Patent Application Number 12/096,531, filed on June 6, 2008.

2. W.K. Cham, C.K. Fong, "*METHODS AND APPARATUS FOR DERIVING AN ORDER-16 INTEGER TRANSFORM*," US Non-Provisional Patent Application Number 12/103,676, filed on April 15, 2008.

3. W.K. Cham, C.K. Fong, "*PROCESSES AND APPARATUS FOR DERIVING ORDER-16 INTEGER TRANSFORMS*," US Non-Provisional Patent Application Number 12/100,358, filed on April 9, 2008.

4. W.K. Cham, C.K. Fong, J. Dong, K.N. Ngan, H.M. Wong, L. Wang , Y. Huo, H.Y. Pun, "*METHOD AND DEVICE FOR ORDER-16 INTEGER TRANSFORM FROM ORDER-8 INTEGER COSINE TRANSFORM*," US Non-Provisional Patent Application Number 11/950,182, filed on December 4, 2007.

5. W.K. Cham, C.K. Fong, "*視頻信號的編碼係數的轉換裝置及其方法*," Chinese Patent Application Number: 200510134530.6.

Acknowledgements

First of all, I would like to express my gratitude towards my supervisor, Prof. Wai-Kuen CHAM, who gave me generous support and invaluable comments and suggestions on my research throughout these years. He provided an excellent working environment with enormous freedom to develop new ideas. His insights and guidance brought me a lot of inspiration. Without his suggestions and advices, many problems I encountered could be difficult to solve. His encouragement and support gave me many chances to attend some coding standard meetings and international conferences. These are very valuable experience in my research life.

In additions, I take this opportunity to thank Prof. King Ngi NGAN, Prof. Hung-Tat TSUI, Prof. Thierry BLU and Prof. Xiaogang WANG of Image and Video Processing Laboratory for providing helpful advices for my studies. I would thank Mr. Yuk-Chung WONG for his maintenance of the computer systems in the laboratory such that we are able to work smoothly. I would like to thank fellow students in our group for making my academic life enjoyable. I am so grateful for their valuable suggestions about my research. It is a pleasure to work and study with them.

Last but not least, I would like to thank my parents, my family and my friends for their support all the time. Without their encouragement and support, I would have never been able to complete this thesis.

Contents

List of Figures

List of Tables

Chapter 1 Introduction

1.1 Introduction to Video Coding

Digital Video Coding has been researched for decades. Researchers have devoted a lot of work to this topic. However, it is still very hot in both research and industrial area. This is because of its extremely high demand. It is getting more and more popular and it relates closely to us. It appears in many applications in our daily life. We watch TV every day. Digital TV broadcast is a typical example of video coding application. Other than TV, many people like to watch movie at home. The movie contents are stored in VCD, DVD or blu-ray disc. They all are storage media of compressed videos. It is very popular to take video with a handheld digital video camcorder or a digital cameras. Compressed videos are stored. People also like to share their own videos with others through World Wide Web. We believe that Youtube [1] is the most famous example. It allows people to share and to distribute video through video streaming over the web. There are thousands of newly uploaded videos and millions of watches in every day. We believe that it is one of the most

important inventions in this decade. As the bandwidth of the mobile network and the processing power of the mobile hand set are improving, video phone call becomes popular. Moreover, many video applications such as surveillance and professional video editing are moving from analog to digital. These all are example of video coding in our daily life. We can see that how frequent we meet with video coding in different area.

Besides the high demand, why we encode or compress videos? Why not store the videos in raw format? This is because the video data is so large that it is almost impossible to be stored in raw format. It wastes too much resource as if we do so. This can be investigated in several aspects:

- **Spatial Resolution**: The spatial resolution of the video frame is increasing, from QCIF (176×144) in old days, to CIF (352×288), WQVGA (416×240), WVGA (832×480) and it becomes High Definition (HD, 720p - 1280×720 and 1080p - 1920×1080) recently. It is expected that it will increase to Ultra High Definition (UHD, 4k × 2k or even larger) in coming future.

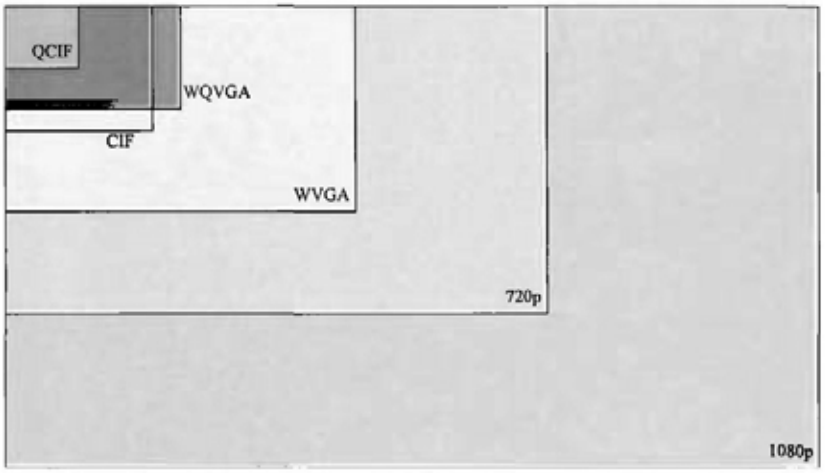


Figure 1-1 A comparison of different frame resolutions.

- **Temporal Sampling Rate**: Interlacing reduces the data rate by half. This is very important when the processing power is limited. As technology is improving, this is no longer the main bottleneck. Progressive video is the main stream nowadays. It is also moving from 25~30 fps to 50~60 fps and even over 100 fps.

- **Color Sub-sampling**: Since human vision system is less sensitive to chroma information, chroma part is usually taken at a lower sampling rate than luma part, for example 4:2:0. This is halved the data rate of 4:4:4 color sub-sampling. As the better picture quality is demanding, higher color sub-sampling rate such as 4:2:2 and even 4:4:4 becomes popular.

- **Color Depth**: Besides higher color sampling rate, higher color depth is also demanding. Conventionally, 8-bit color depth is used. Every pixel in a single color plane is represented using 8 bits. When higher color depth such as 10, 12, 14-bit or higher, more colors can be represented. Picture quality is improved.

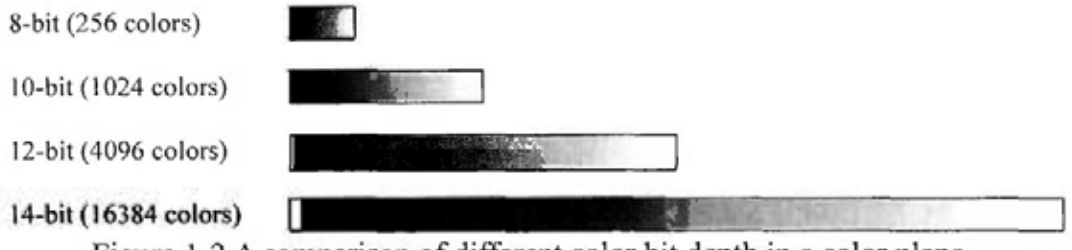


Figure 1-2 A comparison of different color bit depth in a color plane.

Let us take an example here. A typical HD sequence (1920×1080) at 50 fps, with 8-bit color depth, the data rate is 1920×1080×50×3×8 = 2488.32Mbps or 311.04 Mega Byte per second. Although the storage nowadays is very large and low-cost, it is still very inefficient to store raw video. For example a 1 T-Byte (10^{12} Byte, equivalent to

212 4.7G-DVD or 40 single-layered Blu-ray Discs) storage can only store a 3215-second (less than 1-hour) long typical HD sequence. This example shows a typical HD sequence only. A better quality video with higher resolution, higher frame rate and higher sub-sampling rate takes more storage space and requires higher bandwidth to transmit. They can be several times or even several ten times of a typical HD sequence. Video coding significantly reduces the bandwidth required. For example, MPEG-2 has a compression ratio of 1:15 to 1:30 depending on the desired quality. Newer coding standards can offer even higher compression ratio.

It is well-known that our human vision is less sensitive to some kind of distortion, for example, high frequency components. Our human eyes cannot distinguish the difference in high frequency easily. This leads to lossy video coding in most codec designs. Some video contents which are not very sensitive to our human eyes are reduced or even discarded. These distortions are not easily perceived by us. As a result, a higher compression but very small or even almost no visual degradation can be achieved. This is very common in most of the video coding standards.

1.2 Histories of Video Coding Standards

Most of the state-of-the-art video coding standards are based on the generic hybrid video coding model proposed in H.261 [2] twenty years ago. Although it is an old coding standard, many coding tools adopted in this standard are the prototypes of the coding tools nowadays. For example, motion estimation and compensation and transform coding of predicted residue are still fundamental coding tools in the latest video coding standards. Of course, many novel, power and efficient coding tools are

integrated to different standards. Significant performance improvement is offered in every new generation of video coding standards.

- H.261 [2] is an ITU-T (International Telecommunication Union – Telecommunication Standardization Sector) video coding standard issued in 1990. It is designed by the Video Coding Experts Group (VCEG) in the union. Its hybrid video coding framework is a fundamental to many other video coding standards nowadays. Many concepts and ideas in this standard are still doing a good job such as the concept of 16×16 Macroblock (MB), motion compensation, residue coding with DCT, run-length coding and entropy coding. It is an important milestone in video coding technology.

- MPEG-1 [3] is the first international standard including both video and audio specifications. It was developed by Moving Picture Experts Group (MPEG). It is composed of different parts. Its part 2 is the video specification. It is based on the H.261 standard. Its target is to compress video onto a video CD at CIF resolution. The concept of sub-pixel motion compensation was introduced into MPEG-1. It supports up to half pixel accuracy. To enhance the motion estimation and prediction, bi-directionally predicted frame (B-frame) is used. It is predicted with the forward and the backward decoded frames. It reduces the predicted residue significantly and hence the bit rate.

- MPEG-2 [4] is the direct successor of MPEG-1. It was published in 1996. Its Part 2 (also known as H.262) specifies the video coding requirements. It is widely used in DVD standard. The concept of profiles and levels was introduced to MPEG-2 standard. Profile specifies decoding capability in terms

of coding tools while level specifies the constraints on bit rates, frame rates and frame sizes. This makes the decoder manufacturer more flexible to design decoder for one particular application. Motion estimation is also improved. Sub-block level motion estimation is supported such that more precise prediction can be achieved.

- H.263 [5] was proposed by VCEG and released in 1996. It is originally designed for low-bit-rate compression for video conferencing. It has many applications on the internet also.

- MPEG-4 Part 2 (Visual) [6][8] was proposed by MPEG released in 1999. One of the most important features in this standard is object coding. It is possible to code arbitrarily shaped individual video objects. The decoded video object can be moved by the user interactively on the decoder side. The sub-block motion estimation is further improved. The partition sizes allowed includes 8×8, 8×16, 16×8 and 16×16. The motion estimation accuracy can be up to quarter pixel.

- MPEG-4 Part 10 (AVC) (a.k.a. H.264/AVC) [7][8] is a joint projects between ITU-T and ISO/IEC. It was first released in 2003. An amendment called the Fidelity Range Extensions (FRExt) [9] was proposed in 2005. It extended the original standard to provide higher quality video coding. Although MPEG-4 Part 10 is one part of MPEG-4, it is totally different from MPEG-4 Part 2. Object coding in MPEG-4 Part 2 is not present in Part 10. It is the current state-of-the-art AVC standard and it is one of the coding standards to be our testing platform. More information will be provided in later chapter (see Chapter 3).

- Audio Video Standard (AVS) [10] is a multimedia standard proposed by the AVS workgroup in China in 2005. It is not only highly-efficient but also simpler and easier to implement than H.264/AVC. Its coding performance is only slightly lower than H.264/AVC. It is another testing platform in this thesis (see Chapter 4).

- Video Codec 1 (VC-1) [11] is the first video compression algorithm standardized in Society of Motion Picture and Television Engineers (SMPTE) driven by Microsoft released in 2006. It is based on Microsoft Windows Media 9. It supports adaptive block-size transform which supports transform block size of 8×8, 8×4, 4×8 and 4×4. Instead of CAVLC and CABAC in H.264/AVC, VC-1 uses multiple VLC code tables for entropy coding. It also has the fading compensation to tackle the change of brightness level in motion compensation.

Despite of the high performances of the video standards nowadays, work is still being devoted to developing new video coding standards. For example, High Efficiency Video Coding (HEVC, also known as H.265) [12] is being developed by the Joint Collaborative Team on Video Coding (JCT-VC). It is a group of video coding experts from VCEG of ITU-T and MPEG of ISO/IEC. Its target is to further reduce the bit rate for high quality video by half, as compared to H.264/AVC. On the other hand, AVS workgroup in China also started the project for next generation of Audio Video Standard, AVS 2 [14] [15]. AVS2 Ad-hoc group was formed in the 27th AVS workgroup meeting in December 2008. It targets at HD to Super HD resolution sequences. It will support higher color bit depth, higher frame rate and higher color sub-sampling rate. Its FCD will be completed in 2012.

1.3 Generic Hybrid Video Coding

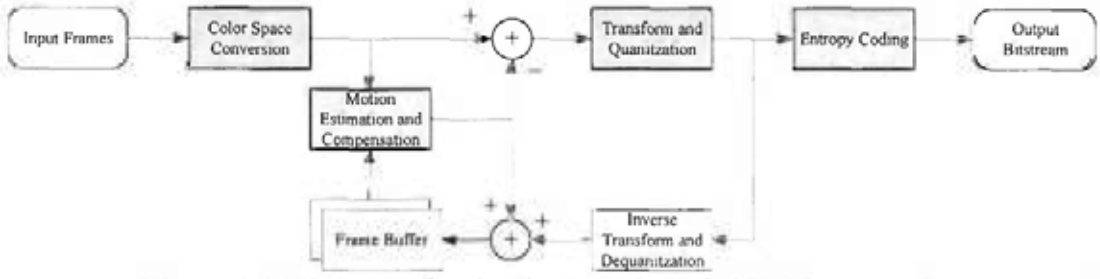


Figure 1-3 Diagram of a simple Generic Hybrid Video Encoder.

Since generic hybrid video encoder is the fundamental and our work is based on this framework, here we are going to give a brief introduction to this encoder for readers who are not familiar to it. Hybrid video coding is a coding algorithm integrating motion compensated inter-picture with spatial transform coding. Generic hybrid encoder is a close-loop encoder which predicts the incoming picture with previously decoded pictures. The encoder can be divided into 4 different functional blocks (in grey) as shown in Figure 1-3. These functional blocks remove different redundancies in the video content.

- **Color space conversion (Color redundancy)**: Color image and video are usually represented by 3 main color components, Red, Green and Blue. It is known as the RGB domain. Each of these components forms a color plane. Picture data in RGB domain usually has high correlation among these color planes. This is the color redundancy. In order to reduce color redundancy, picture data are usually transformed into another color domain which has a lower correlation among its color plane. For example, YUV domain is a common color space used in video coding. Different video coding standards may use

different color domain. However, their aims are the same - to reduce the color redundancy. Pixel data in RGB domain are converted into a specified color space defined in the coding standard before further process, for example, YUV. In (1.1) the RGB to YUV conversion is shown. A color conversion example is shown in Figure 1-4. A color image can be decomposed into the RGB color planes and these planes are converted into YUV planes using (1.1). Y is the luminance while U and V are the chrominance. The variations of the UV planes are smaller than RGB planes and hence the data in UV planes have smaller entropies. Since human vision is less sensitive to the chrominance, UV plane may be further sub-sampled.

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.147 & -0.289 & 0.436 \\ 0.615 & -0.515 & -0.100 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}. \tag{1.1}$$

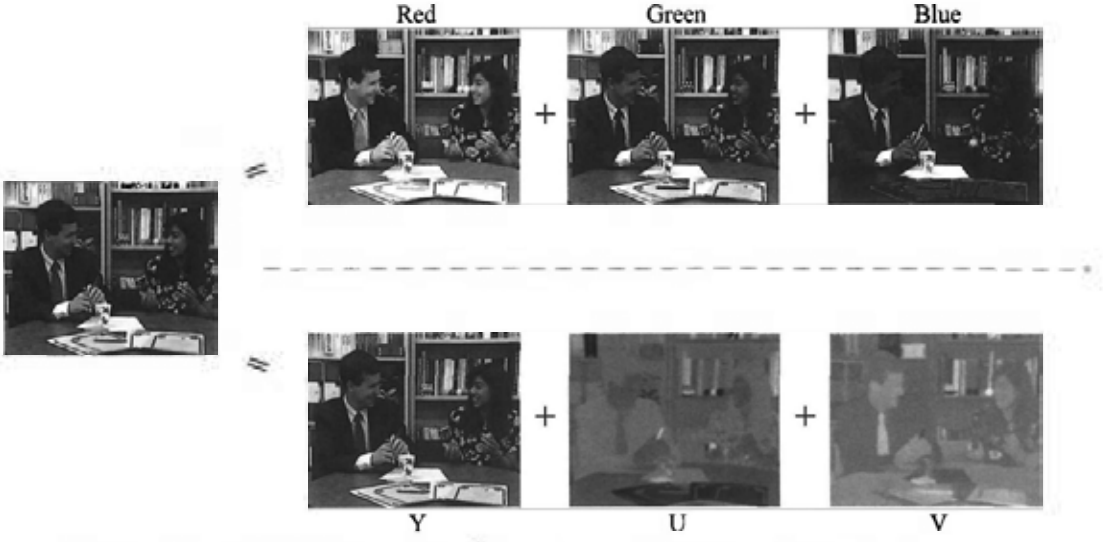


Figure 1-4 An illustration of a given picture in RGB and YUV color space.

- **Motion Estimation and Compensation (Temporal Redundancy)**: It is very trivial that consecutive frames look very similar. The correlations among consecutive frames are very high. This is the temporal redundancy between frames. To remove this redundancy, the incoming frames are predicted with previously decoded frames with motion estimation and compensation. The frame contents are represented by motion vectors (MV) and the predicted residue. This dramatically reduces the video data size.

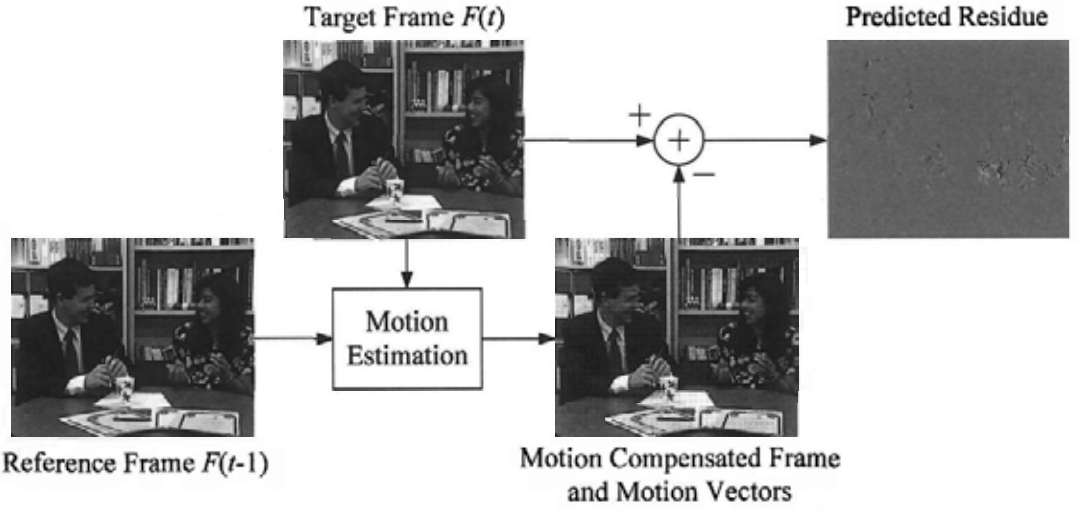
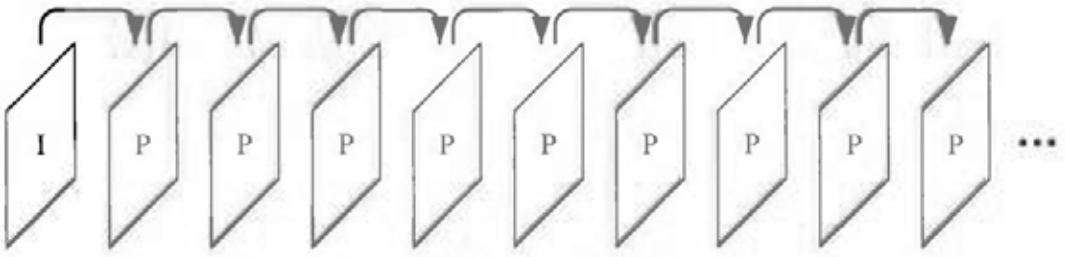
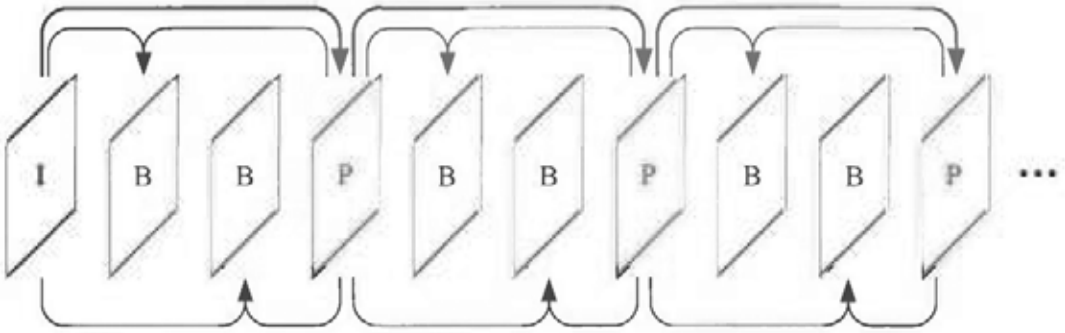


Figure 1-5 An illustration of the motion predicted residue with respect to two consecutive frames.

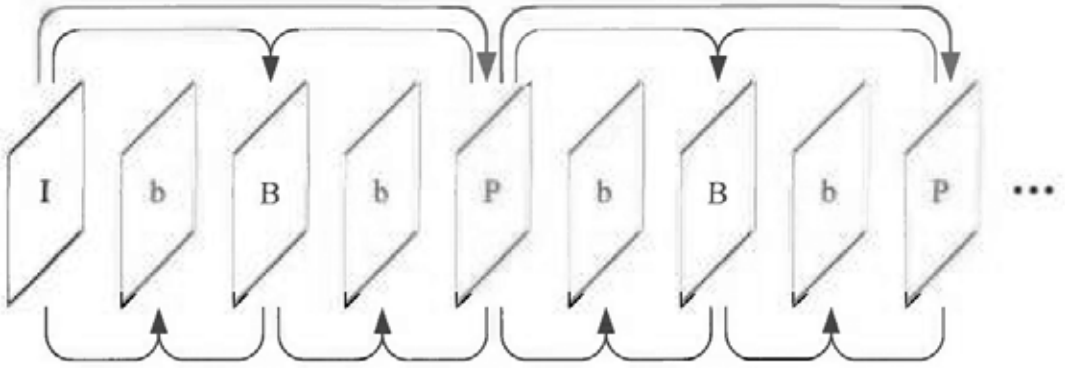
There are several inter-prediction structures. The most trivial one is the IPPP… structure (Figure 1-6(a)) which predict the current frame with previously decoded frame. When bidirectional prediction is allowed, current frame can be predicted by one previous reference frame and one future reference frame. This forms a structure denoted as IBBP… structure (Figure 1-6(b)). This structure can be extended to Hierarchical B-frame structure (Figure 1-6(c)), which is usually denoted as IbBbP….



(a) IPPP… structure



(b) IBBP… structure



(c) Hierarchical B-frame structure

Figure 1-6 Different Prediction Structure. Red arrow is the unidirectional prediction. Blue arrow is the bidirectional prediction.

■ **Transform and Quantization (Spatial Redundancy)**: Although the motion estimation and compensation reduce the video data amount, the data amount of predicted residue can still be reduced. This can be achieved by transform and quantization. Transform decorrelates the residue while quantization reduces or removes the components less sensitive to our human eyes. This lowers the spatial redundancy among the predicted residue.

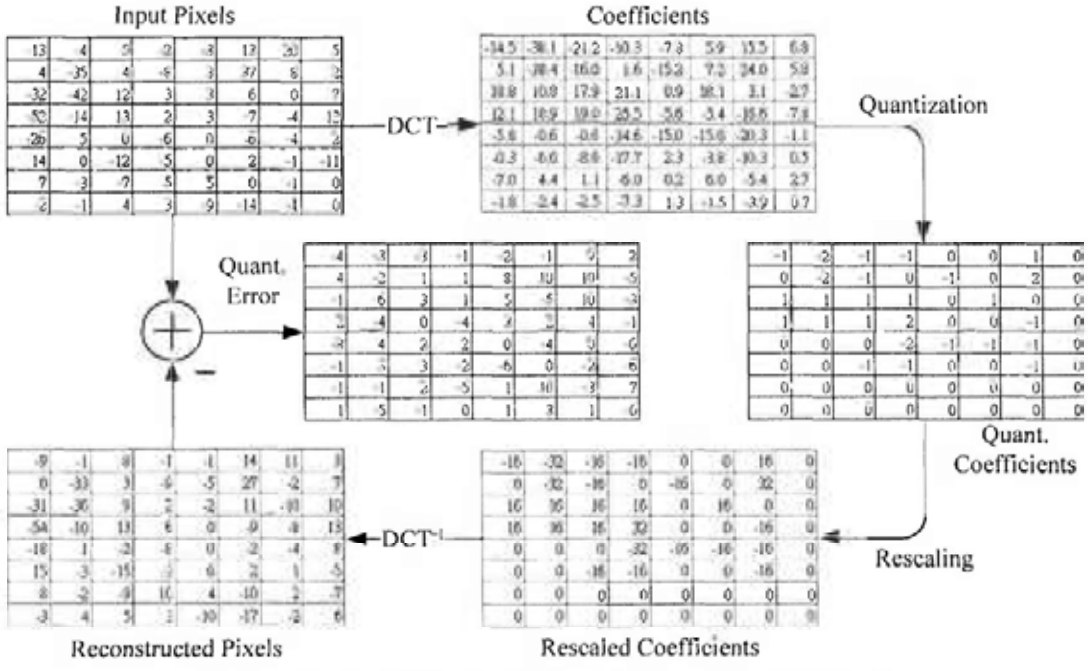


Figure 1-7 An example of transform and quantization.

■ **Entropy Coding (Statistical Redundancy)**: Finally, the transform coefficients in the last step, motion vectors and other side information are coded with a lossless entropy encoder. This encoder gathers the statistics of these contents and assigns the optimal number of bits to the bit stream according to the content and its statistics. This removes the statistical redundancy in the contents.

1.4 Performance Evaluation Metrics

To evaluate the coding performance of a coding system, the bit rate and the PSNR are usually compared. The one offering a higher PSNR at the same bit rate or the one offer the same PSNR at a lower bit rate is a better coding platform or coding algorithm. However, it is very common that they do not align on the same line and this makes the comparison difficult. As an alternative, one may compare the coding performances by plotting the RD curves with several RD points, the PSNR against the bit rates. The one has higher RD curve is a better one. Unfortunately, in some cases, two RD curves may make a cross or both may be too close to each other. It is hard to distinguish which one is the higher one. This makes the comparison difficult. Bjøntegaard proposed a method to calculate the average difference two RD curves in [17]. This is adopted as a common method to evaluate the coding performance in many coding standards, such as H.264/AVC and AVS.

The method proposed in [17] has 3 basic steps. First, a third order polynomial is fitted to 4 RD points (bit rate and PSNR at 4 different QP). Second, obtain an expression for the integral of this curve. Third, the average difference between the curves is calculated as the difference between the integrals divided by the integration interval. The average difference obtained using this method is usually called as BD bit rate and BD PSNR. They are equivalent to each other.

Suppose two coding methods, A and B, are compared and B is the anchor. The RD curves (P_A and P_B) are obtained from 8 RD points (4 from each method) as shown in Figure 1-8.

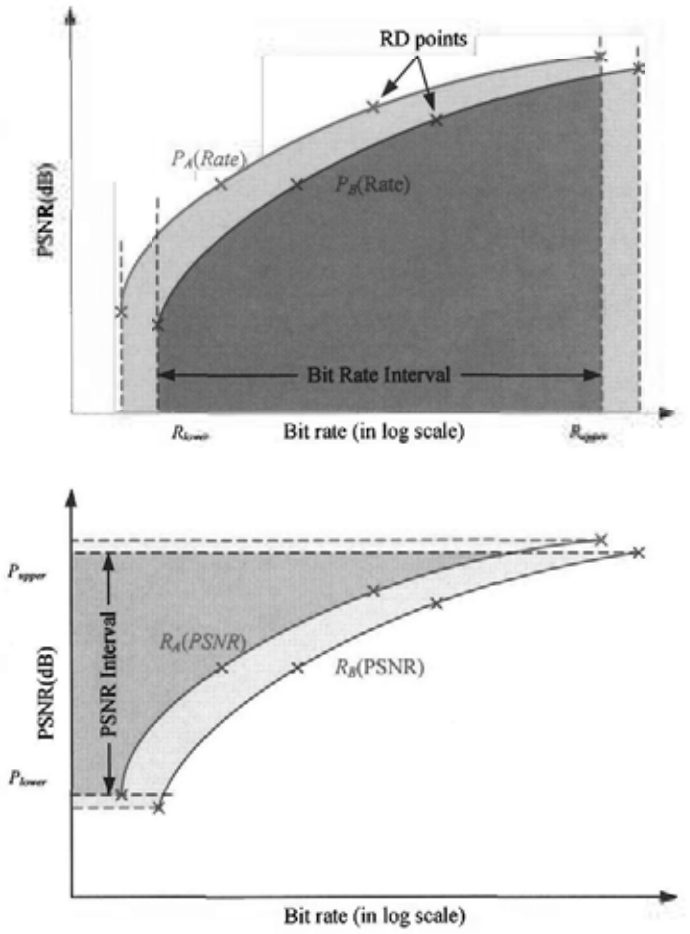


Figure 1-8 Calculation of BD PSNR (upper) and BD bit rate (lower).

P_A and P_B can be expressed in third order polynomials of the rate. BD PSNR is defined as:

$$BD\text{-}PSNR = \frac{1}{R_{upper} - R_{lower}} \int_{R_{lower}}^{R_{upper}} \left(P_A(r) - P_B(r)\right) dr \tag{1.2}$$

Similarly, the curves can be expressed as functions of PSNR and hence the BD bit rate is:

$$BD\text{-}bitrate = \frac{1}{P_{upper} - P_{lower}} \int_{P_{lower}}^{P_{upper}} \left(R_A(p) - R_B(p)\right) dp \tag{1.3}$$

The BD bit rate and BD PSNR are used to measure the performance in this thesis.

1.5 Video Processing in Transform Domain

Besides video coding, there are many video processing applications in transform domain. These applications can perform more efficiently than in pixel domain. On the other hand, the statistics of the transform coefficients plays an important role in many applications.

1.5.1 Fast Walsh Search

Pattern matching in transform domain is an example of video processing in transform domain. An example is the Fast Walsh Search (FWS), which is a pattern match algorithm in Walsh-Hadamard transform domain. It is shown that its accuracy is almost the same as Fast Full Search (FFS) but its complexity is significantly lower than the FFS. FWS is implemented into the H.264/AVC reference software and acts as a fast Motion Estimation (ME) method. It is shown that it outperforms other ME algorithms in the reference software.

1.5.2 Transform Coefficient Distribution

Statistical analysis plays an important role in many video processing applications. Many applications make use of the statistics, especially the distribution, of the data to achieve their works. As a result, the transform coefficient distribution analysis is very important in many processes in transform domain. It is usually assumed that the transform coefficients distribute in Laplace distribution. In our analysis, however, the transform coefficients of intra predicted residue are in Cauchy distribution rather than Laplace one.

1.6 Thesis Scope and Contributions

This thesis can be divided into three parts. The first part is the development of DCT-like integer transform and its application in video coding. The second part is about a fast pattern matching algorithm - Fast Walsh Search. The third part is an analysis of the transform coefficient distribution in predicted residue.

In the first part, one of our contributions is the development of three new integer transforms. We focus on develop order-2^k orthogonal integer transform, especially the order-16 transform. The proposed transforms must have integer kernels. They are going to be integrated into existing video coding systems. They must have good coding performances. Although non-orthogonal transform may also provide a good performance, it is not in our scope. Besides coding performance, efficient computation is also an important factor in designing transform in video coding system. The transform must be separable and have fast algorithm. In this thesis, our aim is to encode sequences with color sub-sampling rate 4:2:0 and MB size of 16×16. We apply order-16 transform to luma part of the predicted residue only. The chroma part is coded with existing 8×8 or 4×4 transforms. However, we expect that order-16 can also be applied to chroma part for sequences with sub-sampling rate 4:4:4 or with support of super macro-block [18]. The proposed transforms are implemented into the reference software of two different coding standards, H.264/AVC and AVS. We focus on the High Profile in H.264/AVC and the Jiaqiang (Enhanced) Profile in AVS. The objective coding performance in these coding standards with different order-16 transforms are measured by the BD

bit rate and BD PSNR described in the previous section. The comparison in terms of subjective picture quality will also be shown. The usage of the order-16 transform is also analyzed. Analysis shows that this usage is pretty high. It reflects the importance of the order-16 transform in video coding.

The second part is the Fast Walsh Search. It is a pattern matching algorithm developed by Mak and Li [19][20][21]. To speed up the matching process, we proposed a statistical threshold to remove the mismatch candidates from the candidate pool. We also analyze the relationship between this threshold and the nature of the pattern. A speed-up of the process without degrading the matching accuracy is shown in the experiment.

The third part is the analysis of the transform coefficient distribution. This is very important to many applications in transform domain and it is usually assume to be in Laplace distribution. However, in our preliminary study, the transform coefficients of the intra-predicted residue are in Cauchy distribution rather than Laplace distribution. We verified this with different video sequences. Although this study is still in a starting stage, it is a very important to many applications such as the transform-based post-processing, the rate control and the rate-distortion optimization (RDO) in video coding.

1.7 Thesis Outlines

In this chapter, a brief introduction about video coding standards has been given. In next chapter, the motivation of our work will be explained. Three different orthogonal order-16 integer transforms will be proposed and demonstrated. Their developments and the abilities will be discussed in detail. They will also be compared with several existing order-16 transforms. These transforms are integrated into the reference software of two different coding standards, H.264/AVC and AVS, in Chapter 3 and Chapter 4 respectively. The implementation details will be clearly discussed. Their coding performances will be compared. Both objective and subjective evaluation will be shown.

In Chapter 5, a new pattern matching method, Fast Walsh Search, will be demonstrated. It is implemented as a motion estimation method into the H.264/AVC reference software. It is shown that its accuracy is similar to Fast Full Search but significantly simpler. It outperforms other fast motion estimation methods in the reference software.

An analysis of the transform coefficient distribution will be presented in Chapter 6. Usually the transform coefficients of images are supposed to be in Laplace distribution. The same assumption is applied to the predicted residue in video coding. However, it is found that the intra-predicted residue is in Cauchy distribution rather than Laplace distribution. Methods to estimate the distribution parameters are also proposed.

1.8 References

[1] Youtube, http://www.youtube.com

[2] ITU-T Recommendation H.261: Video codec for audiovisual services at *p*x64 kbit/s, 1990.

[3] International Standard ISO/IEC 11172-2 MPEG-1 Video, 1993.

[4] International Standard ISO/IEC 13818-2 MPEG-2 Video, 1995.

[5] ITU-T Recommendation H.263: Video coding for low bit rate communication, 1996.

[6] International Standard ISO/IEC 14496-2 MPEG-4 Visual, 2004.

[7] International Standard ISO/IEC 14496-10 MPEG-4 AVC, 2005.

[8] Iain E. G. Richardson, "H.264 and MPEG-4 Video Compression," Wiley, 2003.

[9] D. Marpe, T. Wiegand and S. Gordon, "*H.264/MPEG4-AVC fidelity range extensions: tools, profiles, performance and application areas,*" IEEE ICIP 2005, vol. 1, pp I - 593-596, 2005.

[10] GB/T20090.2 information technology — advanced audio video coding standard Part 2: Video, 2006.

[11] Jay Loomis and Mike Wasson, "*VC-1 Technical Overview,*" October 2007. [Online] Available: http://www.microsoft.com/windows/windowsmedia/howto/articles/vc1techoverview.aspx

[12] HEVC, http://www.vcodex.com/h265.html

[13] JCT-VC, http://www.itu.int/ITU-T/studygroups/com16/jct-vc/

[14] "*Next Generation AVS Video Coding Specification Version 2.0,*" AVS-N1590, March 2009. [Chinese]

[15] "*Next Generation AVS Video Coding - Call for Proposal,*" AVS-N1591, March 2009. [Chinese]

[16] Alois M. Bock, Video Compression Systems – From first principles to concatenated codecs, IET Telecommunications Series 53, 2009.

[17] G. Bjøntegaard, "*Calculation of Average PSNR Differences between RD-curves,*" ITU-T SG16/Q6, Document VCEG-M33, April 2001. [Online] Available:

http://wftp3.itu.int/av-arch/video-site/0104_Aus/

[18] Siwei Ma and C.-C. Jay Kuo, "*High-definition Video Coding with Super-macroblocks*," Proc. SPIE Vol. 6508, 650816, January 2007.

[19] C. M. Mak, N. Li and W. K. Cham, "*Fast motion estimation in Walsh Hadamard domain,*" Proceeding of International Symposium on Intelligent Signal Processing and Communication Systems, pp. 349-352, 2005.

[20] N. Li, C. M. Mak and W. K. Cham, "*Fast block matching algorithm in Walsh-Hadamard domain,*" in Proc. Asian Conference of Computer Vision (ACCV), pp. 712-721, Jan. 2006.

[21] Chun-Man Mak, Chi-Keung Fong and Wai-Kuen Cham, "*Fast Motion Estimation for H.264/AVC in Walsh Hadamard Domain*," IEEE Trans. on CASVT, vol. 18, no. 6, pp. 735-745, June 2008.

Chapter 2 Order-16 DCT-like Integer Transform

2.1 Introduction

Transform coding is a very common and important coding tool in video and image coding. It is a linear process to decorrelate pixel data. Commonly, it is in a block-wise basis. A patch of $N \times N$ pixel data are grouped into a block and this block of data is transformed into $N \times N$ coefficients. Most of the transforms being used in video and image coding are orthogonal and separable. Parseval's Theorem holds when the transform is orthonormal. This means that the total energy in (the sum of square of) the pixel domain is the same as that in transform domain. In addition, the variances of some coefficients are usually larger magnitudes than the others in transform domain. The data are packed into fewer coefficients and hence compression is achieved. In image and video coding, 2-D transform is required. It is important that the transform is separable. If a transform is separable, it means its 2-D transform can be achieved by a column-wise and a row-wise 1-D transform independently. This significantly reduces the complexity from $O(N^4)$ to $O(N^3)$. The

complexity can be further reduced if fast algorithm exists.

Many different transforms for image coding have been proposed, such as Haar transform, Walsh Hadamard Transform (WHT) and Slant transform. The optimal one is Karhunen-Loève Transform (KLT). It is optimally tailored to a group of images. However, it is image-dependent. It depends on the correlation of the image data and hence it is not very popular in image and video coding. Instead, the well-known sub-optimal transform, Discrete Cosine Transform (DCT) [2], is commonly used. It is very popular in many processing and coding applications. This is because the DCT and the KLT are similar when the pixel correlation is approaching to 1. In most natural images, the pixel correlation is very high and approaching to 1. This makes the DCT has a coding performance very close to the optimal KLT. The DCT is also a real, orthogonal and separable transform which can be implemented with real numbers. A number of fast DCT algorithms have been proposed and DCT are widely employed in many signal coding and compression.

Recently, the DCT is replaced by Integer Cosine Transform (ICT) [1]. This is because it also has a very high coding performance similar to the DCT but only required integer arithmetic rather than floating-point. This does not only save the computation dramatically but also reduces the drift error during the transformation. The transform process can be divided into two steps. The first step is a linear transform with integer kernel. The second step is scaling with real numbers. With the integer kernel, its transform coefficients can be computed perfectly with finite bit length. The real scaling constants can be embedded into quantization perfectly. As a result, this solves the problem of drift error. As a result, latest video coding

standards, such as H.264/AVC [4], AVS in China [6] and also VC-1 [7], adopted different ICT.

In image and video coding, fixed block size, says 8×8, DCT was commonly used. However, there is a major drawback in fixed size block based transform. It fails in adapting to varying video signal. This is because the DCT is only sub-optimal to stationary first order Markov random signal. Unfortunately, video signal is changing from time to time and from space to space. As a result, this limits the coding performance. In order to adapt to the non-stationary video signal, Arbitrary Block-size Transform (ABT, also known as Adaptive Block-size Transform and Variable Block-size Transform, VBT) is proposed [8][9]. Transforms of different sizes adapt to different parts of the video picture. Smaller transform codes rapidly changing signals while larger transform codes smoother signals. In H.264/AVC FREXt [10], both 4×4 and 8×8 ICT are adopted to form an ABT system. Either 4×4 or 8×8 ICT is selected to transform the predicted residue in each Macroblock (MB). It is reported that about 10% bit rate reduction is achieved in this ABT system [9]. The addition of smaller transform benefits videos with lower resolution or rich details. Videos with higher resolutions (known as High Definition, HD), such as 1280×720, 1920×1080 or even higher, are becoming more and more popular. It is expected that larger transforms can provide better coding performance to these video sequences. In [19], it is reported that the addition of 16×16 integer transform provides gains from 0.086 to 0.471 dB for HD sequences. As a result, this becomes our motivation to find larger integer transforms (order-16 or above), which have better coding performances and also low computation requirements, to improve the video coding performance.

Besides ABT with larger transforms, many transform based techniques have been proposed to improve the coding performance, such as Shape-Adaptive DCT (SA-DCT) [10][11] in MPEG-4, (Mode Dependent) Directional Transform (MDDT) [12][13]. Despite these two techniques not in our scope, a short introduction will be given.

SA-DCT is based on pre-defined sets of 1-D DCT basis functions. It usually co-operates Video Object (VO) coding in MPEG-4. It transforms the VO data of irregular shape. It is not applicable to all blocks but only to 8×8 blocks with a binary alpha mask boundary that contain one or more transparent pixels. That means it is only applicable to 8×8 blocks along the VO boundary.

In intra-prediction, it is found that the correlation in the predicted residue along the prediction direction is still strong. The statistics of the predicted residues are grouped according to their prediction mode. The 4×4 directional transforms are derived from the KLT according the residue statistics. As a result, different prediction modes take different directional transform. This is known as Mode-Dependent Directional Transform (MDDT) [12]. An alternative is demonstrated in [13]. The directional transforms are selected according the RD performance. Overhead is required to indicate which directional transform is used. This is not only applicable to intra-predicted blocks but also the inter-predicted.

In our work, to seek for order-16 integer transform with better coding performance, we will propose three classes of order-16 transforms in this chapter. The first one is the Simple Integer Transform. It has a simple structure and is derived from order-8 ICT. To improve the performance, the Hybrid Integer Transform from Dyadic Weighted Walsh Transform will be proposed. It is slightly more complex than

Simple Integer Transform. Lastly, a novel algorithm deriving order-16 ICT will be proposed. Two different order-16 ICTs, LLMICT and CSFICT, derived by this algorithm are proposed. Their waveforms are very close to the DCT but implemented in integers only. These proposed transforms aim to improve to coding performance in video coding. They are integrated into the reference software of H.264/AVC and AVS. Their coding performances will be shown in later chapters. Instead, different analysis will be shown in this chapter and it is shown that the proposed transforms have properties similar to the DCT. These transforms also have fast algorithms such that they can be computed efficiently.

Before describing our proposed transforms, let us have a brief description of our symbol notations in the remaining of this chapter first. We are not going to describe the notation of every symbol in detail but we are going to roughly describe how our notation system is. This may help to understand.

Notation	Description
T	Orthogonal transform matrix.
E	The integer kernel of **T**. **E** and **K** form in a pair described in (2.10).
K	The 1-D scaling matrix of **T**. **E** and **K** form in a pair described in (2.10).
S	2-D scaling matrix of **T**. It is a combination of two 1-D scaling matrix **K** described in (2.14)
P	The odd part of **T**.
Q	The even part of **T**.
X	Input pixel data.
F	Transform output coefficients.

Table 2-1 Notation system in later sections

All notations have both subscript and superscript. Their superscript describes their order while their subscript describes their nature or type. For example, $\mathbf{T}_{IC}^{(8)}$ denotes

the order-8 ICT. Its integer kernel and scaling matrix are denoted as $\mathbf{E}_{IC}^{(8)}$ and $\mathbf{K}_{IC}^{(8)}$ respectively.

2.2 The Discrete Cosine Transform

In this thesis, we are only interested in the transforms of order-$N = 2^k$ where k is a positive integer. Without specification, N is restricted to the integer powers of 2 in this thesis. The Discrete Cosine Transform (DCT) referring in this thesis is DCT-II which order-N transform process is defined as:

$$f_j = C_j \sum_{i=0}^{N-1} x_i \cdot \cos\left(\frac{j\pi(2i+1)}{2N}\right) \quad \text{where} \quad C_j = \begin{cases} \sqrt{\frac{1}{N}} & j=0 \\ \sqrt{\frac{2}{N}} & \text{otherwise} \end{cases}, \tag{2.1}$$

x_i and f_j are the input signal and the output transform coefficients respectively. The transform process can be represented in matrix form:

$$\mathbf{F} = \sqrt{\frac{2}{N}} \begin{bmatrix} \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} & \cdots & \sqrt{\frac{1}{2}} \\ \cos\frac{\pi}{2N} & \cos\frac{3\pi}{2N} & \cos\frac{5\pi}{2N} & \cdots & \cos\frac{(2N-1)\pi}{2N} \\ \cos\frac{2\pi}{2N} & \cos\frac{2\cdot 3\pi}{2N} & \cos\frac{2\cdot 5\pi}{2N} & \cdots & \cos\frac{2\cdot(2N-1)\pi}{2N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \cos\frac{(N-1)\pi}{2N} & \cos\frac{(N-1)\cdot 3\pi}{2N} & \cos\frac{(N-1)\cdot 5\pi}{2N} & \cdots & \cos\frac{(N-1)\cdot(2N-1)\pi}{2N} \end{bmatrix} \mathbf{X} = \mathbf{T}_{DCT}^{(N)} \mathbf{X}. \tag{2.2}$$

$\mathbf{T}_{DCT}^{(N)}$ is an $N \times N$ orthogonal matrix such that:

$$\mathbf{T}_{DCT}^{(N)} \, \mathbf{T}_{DCT}^{(N)\,T} = \mathbf{I}_N \quad \text{or equivalent to} \quad \mathbf{T}_{DCT}^{(N)\,T} = \left(\mathbf{T}_{DCT}^{(N)}\right)^{-1}, \tag{2.3}$$

$\mathbf{I}_N$ is an order-N identity matrix. The (i, j)th element in $\mathbf{T}_{DCT}^{(N)}$ is equal to:

$$T_{DCT}^{(N)}(i, j) = C_j \cos\left(\frac{\pi(2i-1)j}{2N}\right) \quad \text{where} \quad C_j = \begin{cases} \sqrt{\frac{1}{N}} & j=0 \\ \sqrt{\frac{2}{N}} & \text{otherwise} \end{cases}. \tag{2.4}$$

Due to the dyadic symmetry, an order-$2N$ DCT can be decomposed into the odd part $\mathbf{P}_{DCT}^{(N)}$ and the even part $\mathbf{Q}_{DCT}^{(N)}$ [23][24][25]:

$$\mathbf{T}_{DCT}^{(2N)} = \begin{bmatrix} \mathbf{Q}_{DCT}^{(N)} & \mathbf{0}_N \\ \mathbf{0}_N & \mathbf{P}_{DCT}^{(N)} \end{bmatrix} \begin{bmatrix} \mathbf{I}_N & \tilde{\mathbf{I}}_N \\ \mathbf{I}_N & -\tilde{\mathbf{I}}_N \end{bmatrix} \quad \text{for } N \geq 2. \tag{2.5}$$

$\tilde{\mathbf{I}}_N$ is $\mathbf{I}_N$ rotated by 90° while $\mathbf{0}_N$ is an order-N zero matrix. It is proven that $\mathbf{Q}_{DCT}^{(N)}$ is a scaled version of the DCT

$$\mathbf{Q}_{DCT}^{(N)} = \frac{1}{\sqrt{2}} \mathbf{T}_{DCT}^{(N)} \quad \text{for } N \geq 2. \tag{2.6}$$

As a result, the DCT has a recursive structure:

$$\mathbf{T}_{DCT}^{(2N)} = \begin{bmatrix} \mathbf{T}_{DCT}^{(N)} & \mathbf{0}_N \\ \mathbf{0}_N & \mathbf{P}_{DCT}^{(N)} \end{bmatrix} \begin{bmatrix} \mathbf{I}_N & \tilde{\mathbf{I}}_N \\ \mathbf{I}_N & -\tilde{\mathbf{I}}_N \end{bmatrix} \quad \text{for } N \geq 2. \tag{2.7}$$

The elements in $\mathbf{T}_{DCT}^{(N)}$ appear more than once and hence $\mathbf{T}_{DCT}^{(N)}$ can be represented with a few different elements a_i. The elements with the same magnitude are represented with the same a_i. We can generalize a_i such that it can be any real numbers which keep the matrix orthogonal with the structure same as $\mathbf{T}_{DCT}^{(N)}$. In this thesis, this transform is called General Cosine Transform (GCT). For example, the order-8 GCT is shown in (2.8). Obviously, order-8 DCT is an example of GCT with its elements $\{a_0, a_1 \ldots a_6\} = \{\frac{1}{\sqrt{8}}, \frac{1}{2}\cos(\frac{\pi}{16}), \frac{1}{2}\cos(\frac{3\pi}{16}), \frac{1}{2}\cos(\frac{5\pi}{16}), \frac{1}{2}\cos(\frac{7\pi}{16}), \frac{1}{2}\cos(\frac{\pi}{8}), \frac{1}{2}\cos(\frac{3\pi}{8})\}$.

$$\mathbf{T}_{GC}^{(8)} = \begin{bmatrix} a_0 & a_0 & a_0 & a_0 & a_0 & a_0 & a_0 & a_0 \\ a_1 & a_2 & a_3 & a_4 & -a_4 & -a_3 & -a_2 & -a_1 \\ a_5 & a_6 & -a_6 & -a_5 & -a_5 & -a_6 & a_6 & a_5 \\ a_2 & -a_4 & -a_1 & -a_3 & a_3 & a_1 & a_4 & -a_2 \\ a_0 & -a_0 & -a_0 & a_0 & a_0 & -a_0 & -a_0 & a_0 \\ a_3 & -a_1 & a_4 & a_2 & -a_2 & -a_4 & a_1 & -a_3 \\ a_6 & -a_5 & a_5 & -a_6 & -a_6 & a_5 & -a_5 & a_6 \\ a_4 & -a_3 & a_2 & -a_1 & a_1 & -a_2 & a_3 & -a_4 \end{bmatrix}, \tag{2.8}$$

GCT has properties same as the DCT. (2.7) can be rewritten as:

$$\mathbf{T}_{GC}^{(2N)} = \begin{bmatrix} \mathbf{Q}_{GC}^{(N)} & \mathbf{0}_N \\ \mathbf{0}_N & \mathbf{P}_{GC}^{(N)} \end{bmatrix} \begin{bmatrix} \mathbf{I}_N & \tilde{\mathbf{I}}_N \\ \mathbf{I}_N & -\tilde{\mathbf{I}}_N \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}}\mathbf{T}_{GC}^{(N)} & \mathbf{0}_N \\ \mathbf{0}_N & \mathbf{P}_{GC}^{(N)} \end{bmatrix} \begin{bmatrix} \mathbf{I}_N & \tilde{\mathbf{I}}_N \\ \mathbf{I}_N & -\tilde{\mathbf{I}}_N \end{bmatrix} \text{ for } N \geq 2. \tag{2.9}$$

This is an important property to derive order-16 integer transforms in later sections. $\mathbf{T}_{GC}^{(2N)}$ is orthogonal only when both $\mathbf{T}_{GC}^{(N)}$ and $\mathbf{P}_{GC}^{(N)}$ are orthogonal.

2.3 Integer Cosine Transform

2.3.1 Order-4 and Order-8 ICT

Order-8 integer cosine transform (ICT) in existing coding systems was first proposed in [1]. It is defined that an integer transform **T** is an orthogonal matrix which can be composed by a scaling matrix **K** and an integer kernel **E**.

$$\mathbf{T} = \mathbf{K}\,\mathbf{E}, \tag{2.10}$$

E is a matrix which contains only integer elements and **K** is a diagonal matrix with positive real elements, which makes the basis vectors of **T** unity. An ICT has both the properties of an integer transform and a GCT. It is a GCT which has an integer kernel:

$$\mathbf{T}_{IC}^{(N)} = \mathbf{K}_{IC}^{(N)} \mathbf{E}_{IC}^{(N)}. \tag{2.11}$$

The structure of $\mathbf{E}_{IC}^{(N)}$ is the same as $\mathbf{T}_{GC}^{(N)}$ but $\mathbf{E}_{IC}^{(N)}$ contains only integers. It also has a recursive structure:

$$\mathbf{T}_{IC}^{(2N)} = \begin{bmatrix} \frac{1}{\sqrt{2}}\mathbf{T}_{IC}^{(N)} & \mathbf{0}_N \\ \mathbf{0}_N & \mathbf{P}_{IC}^{(N)} \end{bmatrix} \begin{bmatrix} \mathbf{I}_N & \tilde{\mathbf{I}}_N \\ \mathbf{I}_N & -\tilde{\mathbf{I}}_N \end{bmatrix} \text{ for } N \geq 2. \tag{2.12}$$

Order-4 ICT, $\mathbf{T}_{IC}^{(4)}$, is defined as:

$$\mathbf{T}_{IC}^{(4)} = \begin{bmatrix} a_0 & a_0 & a_0 & a_0 \\ a_1 & a_2 & -a_2 & -a_1 \\ a_0 & -a_0 & -a_0 & a_0 \\ a_2 & -a_1 & a_1 & -a_2 \end{bmatrix}. \tag{2.13}$$

It is orthogonal for any (a_0, a_1, a_2). $\text{ICT}_4(1, 2, 1)$ and $\text{ICT}_4(2, 3, 1)$ are adopted in H.264 and AVS respectively. For order-8 ICT, $\mathbf{T}_{IC}^{(8)}$ is:

$$\mathbf{T}_{IC}^{(8)} = \begin{bmatrix} a_0 & a_0 & a_0 & a_0 & a_0 & a_0 & a_0 & a_0 \\ a_1 & a_2 & a_3 & a_4 & -a_4 & -a_3 & -a_2 & -a_1 \\ a_5 & a_6 & -a_6 & -a_5 & -a_5 & -a_6 & a_6 & a_5 \\ a_2 & -a_4 & -a_1 & -a_3 & a_3 & a_1 & a_4 & -a_2 \\ a_0 & -a_0 & -a_0 & a_0 & a_0 & -a_0 & -a_0 & a_0 \\ a_3 & -a_1 & a_4 & a_2 & -a_2 & -a_4 & a_1 & -a_3 \\ a_6 & -a_5 & a_5 & -a_6 & -a_6 & a_5 & -a_5 & a_6 \\ a_4 & -a_3 & a_2 & -a_1 & a_1 & -a_2 & a_3 & -a_4 \end{bmatrix}. \tag{2.14}$$

It can be decomposed into:

$$\mathbf{T}_{IC}^{(8)} = \begin{bmatrix} \mathbf{Q}_{IC}^{(4)} & \mathbf{0}_4 \\ \mathbf{0}_4 & \mathbf{P}_{IC}^{(4)} \end{bmatrix} \begin{bmatrix} \mathbf{I}_4 & \tilde{\mathbf{I}}_4 \\ \mathbf{I}_4 & -\tilde{\mathbf{I}}_4 \end{bmatrix} \tag{2.15}$$

where

$$\mathbf{Q}_{IC}^{(4)} = \begin{bmatrix} a_0 & a_0 & a_0 & a_0 \\ a_5 & a_6 & -a_6 & -a_5 \\ a_0 & -a_0 & -a_0 & a_0 \\ a_6 & -a_5 & a_5 & -a_6 \end{bmatrix} \text{ is a scaled version of } \mathbf{T}_{IC}^{(4)} \text{ and} \tag{2.16}$$

$$\mathbf{P}_{IC}^{(4)} = \begin{bmatrix} a_1 & a_2 & a_3 & a_4 \\ a_2 & a_4 & -a_1 & -a_3 \\ a_3 & -a_1 & -a_4 & a_2 \\ a_4 & -a_3 & a_2 & -a_1 \end{bmatrix}. \tag{2.17}$$

$\mathbf{Q}_{IC}^{(4)}$ is orthogonal for any a_0, a_5 and a_6. $\mathbf{P}_{IC}^{(4)}$ is orthogonal if:

$$a_1 a_2 = a_1 a_3 + a_2 a_4 + a_3 a_4 \tag{2.18}$$

so as to ensure that $\mathbf{T}_{IC}^{(8)}$ is orthogonal. To make the basis vectors have waveforms similar to the DCT:

$$a_1 \geq a_2 \geq a_3 \geq a_4 \text{ and } a_5 \geq a_6. \tag{2.19}$$

The 2-D transform process of an $N \times N$ data block $\mathbf{X}$ is modeled as:

$$\mathbf{F} = \mathbf{T}_{IC}^{(N)} \, \mathbf{X} \, \mathbf{T}_{IC}^{(N)^T}. \tag{2.20}$$

Substituting (2.11) into (2.20) to yield:

$$\begin{aligned} \mathbf{F} &= \mathbf{K}_{IC}^{(N)} \mathbf{E}_{IC}^{(N)} \, \mathbf{X} \, \mathbf{E}_{IC}^{(N)^T} \mathbf{K}_{IC}^{(N)} \\ &= \mathbf{S}_{IC}^{(N)} \odot \left(\mathbf{E}_{IC}^{(N)} \, \mathbf{X} \, \mathbf{E}_{IC}^{(N)^T} \right) \end{aligned} \tag{2.21}$$

$\mathbf{S}_{IC}^{(N)}$ is the 2-D scale matrix formed by the 1-D diagonal scaling matrix $\mathbf{K}_{IC}^{(N)}$. The element-to-element multiplication is represented by $\odot$. $\mathbf{S}_{IC}^{(N)}$ is defined as:

$$S_{IC}^{N}(i,j) = K_{IC}^{N}(i,i) \times K_{IC}^{N}(j,j). \tag{2.22}$$

The inverse transform can be implemented as:

$$\mathbf{X} = \mathbf{E}_{IC}^{(N)^T}\left(\mathbf{F} \odot \mathbf{S}_{IC}^{(N)}\right)\mathbf{E}_{IC}^{(N)}. \tag{2.23}$$

In usual manner, the 2-D scaling matrix of an integer transform, **S**, is integrated into the quantization and rescaling process in coding system. The transform with integer kernel, **E**, can be perfectly implemented with finite bit length. The drifting error caused by irrational transform matrix (such as the DCT) can be prevented. ICT also has this advantage.

There are infinite many integer solutions that satisfy (2.18). As a result, there are infinite possible cases for $\mathbf{T}_{IC}^{(8)}$. However, finite sets of a_i, which obtain higher transform efficiency (See Section 2.8.2 for detail), were suggested in [1]. The higher transform efficiency means the higher ability to decorrelate the data. This also implies higher compression ability. In the following, we name the order-N $\mathbf{T}_{IC}^{(N)}$ as $\mathrm{ICT}_N(a_0, a_1\ldots)$ for ease of use. The larger value range of a_i, the better performance can be obtained. The larger value range of a_i also means higher precision and higher complexity usually. To balance the compression ability and the complexity, two sets of a_i, $\mathrm{ICT}_8(8, 12, 10, 6, 3, 8, 4)$ and $\mathrm{ICT}_8(8, 10, 9, 6, 2, 10, 4)$, are chosen. They are adopted in the latest video coding standards – H.264/AVC FREXt [5] and AVS [6] respectively. In H.264/AVC FREXt, order-4 ICT is also used.

2.3.2 Order-16 ICT

Order-16 ICT was proposed in [14] and [15]. Its integer kernel, $\mathbf{E}_{IC}^{(16)}$, is defined as:

$$\mathbf{T}_{IC}^{(16)} = \left[\begin{array}{cccccccc|cccc} a_0 & a_0 & a_0 & a_0 & a_0 & a_0 & a_0 & a_0 & a_0 & a_0 & \cdots & a_0 \\ a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7 & a_8 & -a_8 & -a_7 & \cdots & -a_1 \\ a_9 & a_{10} & a_{11} & a_{12} & -a_{12} & -a_{11} & -a_{10} & -a_9 & -a_9 & -a_{10} & \cdots & a_9 \\ a_2 & a_5 & a_8 & -a_6 & -a_3 & -a_1 & -a_4 & -a_7 & a_7 & a_4 & \cdots & -a_2 \\ a_{13} & a_{14} & -a_{14} & -a_{13} & -a_{13} & -a_{14} & a_{14} & a_{13} & a_{13} & a_{14} & \cdots & a_{13} \\ a_3 & a_8 & -a_4 & -a_2 & -a_7 & a_5 & a_1 & a_6 & -a_6 & -a_1 & \cdots & -a_3 \\ a_{10} & -a_{12} & -a_9 & -a_{11} & a_{11} & a_9 & a_{12} & -a_{10} & -a_{10} & a_{12} & \cdots & a_{10} \\ a_4 & -a_6 & -a_2 & a_8 & a_1 & a_7 & -a_3 & -a_5 & a_5 & a_3 & \cdots & -a_4 \\ a_0 & -a_0 & -a_0 & a_0 & a_0 & -a_0 & -a_0 & a_0 & a_0 & -a_0 & \cdots & a_0 \\ a_5 & -a_3 & -a_7 & a_1 & -a_8 & -a_2 & a_6 & a_4 & -a_4 & -a_6 & \cdots & -a_5 \\ a_{11} & -a_9 & a_{12} & a_{10} & -a_{10} & -a_{12} & a_9 & -a_{11} & -a_{11} & a_9 & \cdots & a_{11} \\ a_6 & -a_1 & a_5 & a_7 & -a_2 & a_4 & a_8 & -a_3 & a_3 & -a_8 & \cdots & -a_6 \\ a_{14} & -a_{13} & a_{13} & -a_{14} & -a_{14} & a_{13} & -a_{13} & a_{14} & a_{14} & -a_{13} & \cdots & a_{14} \\ a_7 & -a_4 & a_1 & -a_3 & a_1 & a_8 & -a_5 & a_2 & -a_2 & a_5 & \cdots & -a_7 \\ a_{12} & -a_{11} & a_{10} & -a_9 & a_1 & -a_{10} & a_{11} & -a_{12} & -a_{12} & a_{11} & \cdots & a_{12} \\ a_8 & -a_7 & a_6 & -a_5 & a_4 & -a_3 & a_2 & -a_1 & a_1 & -a_2 & \cdots & -a_8 \end{array}\right], \tag{2.24}$$

Similar to order-8 ICT, it can be decomposed into:

$$\mathbf{T}_{IC}^{(16)} = \begin{bmatrix} \frac{1}{\sqrt{2}}\mathbf{T}_{IC}^{(8)} & \mathbf{0}_8 \\ \mathbf{0}_8 & \mathbf{P}_{IC}^{(8)} \end{bmatrix} \begin{bmatrix} \mathbf{I}_8 & \widetilde{\mathbf{I}}_8 \\ \mathbf{I}_8 & -\widetilde{\mathbf{I}}_8 \end{bmatrix}. \tag{2.25}$$

$\mathbf{P}_{IC}^{(8)}$ is the odd part of $\mathbf{T}_{IC}^{(16)}$. It has a structure:

$$\mathbf{P}_{IC}^{(16)} = \begin{bmatrix} a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7 & a_8 \\ a_2 & a_5 & a_8 & -a_6 & -a_3 & -a_1 & -a_4 & -a_7 \\ a_3 & a_8 & -a_4 & -a_2 & -a_7 & a_5 & a_1 & a_6 \\ a_4 & -a_6 & -a_2 & a_8 & a_1 & a_7 & -a_3 & -a_5 \\ a_5 & -a_3 & -a_7 & a_1 & -a_8 & -a_2 & a_6 & a_4 \\ a_6 & -a_1 & a_5 & a_7 & -a_2 & a_4 & a_8 & -a_3 \\ a_7 & -a_4 & a_1 & -a_3 & a_6 & a_8 & -a_5 & a_2 \\ a_8 & -a_7 & a_6 & -a_5 & a_4 & -a_3 & a_2 & -a_1 \end{bmatrix}. \tag{2.26}$$

To ensure $\mathbf{T}_{IC}^{(16)}$ orthogonal, both $\mathbf{T}_{IC}^{(8)}$ and $\mathbf{P}_{IC}^{(8)}$ must be orthogonal. The criteria for orthogonality are:

$$a_1a_2 + a_2a_5 + a_3a_8 = a_4a_6 + a_3a_5 + a_1a_6 + a_4a_7 + a_7a_8, \tag{2.27}$$

$$a_1a_3 + a_2a_8 + a_5a_6 + a_1a_7 + a_6a_8 = a_3a_4 + a_2a_4 + a_5a_7, \tag{2.28}$$

$$a_1a_4 + a_1a_5 + a_4a_8 + a_6a_7 = a_2a_6 + a_2a_3 + a_3a_7 + a_5a_8, \text{ and} \tag{2.29}$$

$$a_9a_{10} = a_9a_{11} + a_{10}a_{12} + a_{11}a_{12}. \tag{2.30}$$

Some solutions to above equations are suggested in [14] and [15]. However, it is not easy to figure out the fast algorithm for $\mathbf{P}_{IC}^{(8)}$ and hence that for $\mathbf{T}_{IC}^{(16)}$. As a result, there is no fast algorithm proposed for order-16 ICT. This motivates us to develop other high performance order-16 orthogonal integer transforms with fast algorithms.

2.3.3 Other Order-16 Integer Transforms

There are a number of order-16 integer transform [15]-[18] proposed in recent years. The integer kernels of these proposed order-16 transforms are listed in (2.31) to (2.34) respectively. With these integer kernels, their transform can be easily found according to (2.10).

$$\mathbf{E}_{Wien} = \begin{bmatrix}
17 & 17 & 17 & 17 & 17 & 17 & 17 & 17 & 17 & 17 & 17 & 17 & 17 & 17 & 17 & 17 \\
22 & 28 & 12 & 20 & 12 & 16 & 8 & 6 & -6 & -8 & -16 & -12 & -20 & -12 & -28 & -22 \\
24 & 20 & 12 & 6 & -6 & -12 & -20 & -24 & -24 & -20 & -12 & -6 & 6 & 12 & 20 & 24 \\
28 & 12 & 6 & -16 & -12 & -22 & -20 & -8 & 8 & 20 & 22 & 12 & 16 & -6 & -12 & -28 \\
23 & 7 & -7 & -23 & -23 & -7 & 7 & 23 & 23 & 7 & -7 & -23 & -23 & -7 & 7 & 23 \\
12 & 6 & -20 & -28 & -8 & 12 & 22 & 16 & -16 & -22 & -12 & 8 & 28 & 20 & -6 & -12 \\
20 & -6 & -24 & -12 & 12 & 24 & 6 & -20 & -20 & 6 & 24 & 12 & -12 & -24 & -6 & 20 \\
20 & -16 & -28 & 6 & 22 & 8 & -12 & -12 & 12 & 12 & -8 & -22 & -6 & 28 & 16 & -20 \\
17 & -17 & -17 & 17 & 17 & -17 & -17 & 17 & 17 & -17 & -17 & 17 & 17 & -17 & -17 & 17 \\
12 & -12 & -8 & 22 & -6 & -28 & 16 & 20 & -20 & -16 & 28 & 6 & -22 & 8 & 12 & -12 \\
12 & -24 & 6 & 20 & -20 & -6 & 24 & -12 & -12 & 24 & -6 & -20 & 20 & 6 & -24 & 13 \\
16 & -22 & 12 & 8 & -28 & 20 & 6 & -12 & 12 & -6 & -20 & 28 & -8 & -12 & 22 & -16 \\
7 & -23 & 23 & -7 & -7 & 23 & -23 & 7 & 7 & -23 & 23 & -7 & -7 & 23 & -23 & 7 \\
8 & -20 & 22 & -12 & 16 & 6 & -12 & 28 & -28 & 12 & -6 & -16 & 12 & -22 & 20 & -8 \\
6 & -12 & 20 & -24 & 24 & -20 & 12 & -6 & -6 & 12 & -20 & 24 & -24 & 20 & -12 & 6 \\
6 & -8 & 16 & -12 & 20 & -12 & 28 & -22 & 22 & -28 & 12 & -20 & 12 & -16 & 8 & -6
\end{bmatrix}, \tag{2.31}$$

$$\mathbf{E}_{Chen} = \begin{bmatrix}
32 & 32 & 32 & 32 & 32 & 32 & 32 & 32 & 32 & 32 & 32 & 32 & 32 & 32 & 32 & 32 \\
45 & 43 & 40 & 35 & 29 & 21 & 13 & 4 & -4 & -13 & -21 & -29 & -35 & -40 & -43 & -45 \\
44 & 38 & 25 & 9 & -9 & -25 & -38 & -44 & -44 & -38 & -25 & -9 & 9 & 25 & 38 & 44 \\
43 & 29 & 4 & -21 & -40 & -45 & -35 & -13 & 13 & 35 & 45 & 40 & 21 & -4 & -29 & -43 \\
42 & 17 & -17 & -42 & -42 & -17 & 17 & 42 & 42 & 17 & -17 & -42 & -42 & -17 & 17 & 42 \\
40 & 4 & -35 & -43 & -13 & 29 & 45 & 21 & -21 & -45 & -29 & 13 & 43 & 35 & -4 & -40 \\
38 & -9 & -44 & -25 & 25 & 44 & 9 & -38 & -38 & 9 & 44 & 25 & -25 & -44 & -9 & 38 \\
35 & -21 & -43 & 4 & 45 & 13 & -40 & -29 & 29 & 40 & -13 & -45 & -4 & 43 & 21 & -35 \\
32 & -32 & -32 & 32 & 32 & -32 & -32 & 32 & 32 & -32 & -32 & 32 & 32 & -32 & -32 & 32 \\
29 & -40 & -13 & 45 & -4 & -43 & 21 & 35 & -35 & -21 & 43 & 4 & -45 & 13 & 40 & -29 \\
25 & -44 & 9 & 38 & -38 & -9 & 44 & -25 & -25 & 44 & -9 & -38 & 38 & 9 & -44 & 25 \\
21 & -45 & 29 & 13 & -43 & 35 & 4 & -40 & 40 & -4 & -35 & 43 & -13 & -29 & 45 & -21 \\
17 & -42 & 42 & -17 & -17 & 42 & -42 & 17 & 17 & -42 & 42 & -17 & -17 & 42 & -42 & 17 \\
13 & -35 & 45 & -40 & 21 & 4 & -29 & 43 & -43 & 29 & -4 & -21 & 40 & -45 & 35 & -13 \\
9 & -25 & 38 & -44 & 44 & -38 & 25 & -9 & -9 & 25 & -38 & 44 & -44 & 38 & -25 & 9 \\
4 & -13 & 21 & -29 & 35 & -40 & 43 & -45 & 45 & -43 & 40 & -35 & 29 & -21 & 13 & -4
\end{bmatrix} \tag{2.32}$$

$$\mathbf{E}_{Lee} = \begin{bmatrix}
16 & 16 & 16 & 16 & 16 & 16 & 16 & 16 & 16 & 16 & 16 & 16 & 16 & 16 & 16 & 16 \\
27 & 25 & 28 & 21 & 21 & 9 & 11 & 3 & -3 & -11 & -9 & -21 & -21 & -28 & -25 & -27 \\
25 & 21 & 14 & 5 & -5 & -14 & -21 & -25 & -25 & -21 & -14 & -5 & 5 & 14 & 21 & 25 \\
25 & 21 & 3 & -9 & -28 & -27 & -21 & -11 & 11 & 21 & 27 & 28 & 9 & -3 & -21 & -25 \\
17 & 7 & -7 & -17 & -17 & -7 & 7 & 17 & 17 & 7 & -7 & -17 & -17 & -7 & 7 & 17 \\
28 & 3 & -21 & -25 & -11 & 21 & 27 & 9 & -9 & -27 & -21 & 11 & 25 & 21 & -3 & -28 \\
21 & -5 & -25 & -14 & 14 & 25 & -5 & -21 & -21 & 5 & 25 & 14 & -14 & -25 & -5 & 21 \\
21 & -9 & -25 & 3 & 27 & 11 & -28 & -21 & 16 & 28 & -11 & -27 & -3 & 25 & -9 & -21 \\
16 & -16 & -16 & 16 & 16 & -16 & -16 & 16 & 16 & -16 & -16 & 16 & 16 & -16 & -16 & 16 \\
21 & -28 & -11 & 27 & -3 & -25 & 9 & 21 & -21 & -9 & 25 & 3 & -27 & 11 & 28 & -21 \\
14 & -25 & 5 & 21 & -21 & -5 & 25 & -14 & -14 & 25 & -5 & -21 & 21 & 5 & -25 & 14 \\
9 & -27 & 21 & 11 & -25 & 21 & 3 & -28 & 28 & -3 & -21 & 25 & -11 & -21 & 27 & -9 \\
7 & -17 & 17 & -7 & -7 & 17 & -17 & 7 & 7 & -17 & 17 & -7 & -7 & 17 & -17 & 7 \\
11 & -21 & 27 & -28 & 9 & 3 & -21 & 25 & -25 & 21 & -3 & -9 & 28 & -27 & 21 & -11 \\
5 & -14 & 21 & -25 & 25 & -21 & 14 & -5 & -5 & 14 & -21 & 25 & -25 & 21 & -14 & 5 \\
3 & -11 & 9 & -21 & 21 & -28 & 25 & -27 & 27 & -25 & 28 & -21 & 21 & -9 & 11 & -3
\end{bmatrix} \tag{2.33}$$

$$\mathbf{E}_{Jridhi} = \begin{bmatrix}
256 & 256 & 256 & 256 & 256 & 256 & 256 & 256 & 256 & 256 & 256 & 256 & 256 & 256 & 256 & 256 \\
344 & 336 & 304 & 272 & 216 & 168 & 88 & 48 & -48 & -88 & -168 & -216 & -272 & -304 & -336 & -344 \\
216 & 184 & 120 & 40 & -40 & -120 & -184 & -216 & -216 & -184 & -120 & -40 & 40 & 120 & 184 & 216 \\
227 & 139 & 29 & -116 & -203 & -232 & -188 & -56 & 56 & 188 & 232 & 203 & 116 & -29 & -139 & -227 \\
320 & 128 & -128 & -320 & -320 & -128 & 128 & 320 & 320 & 128 & -128 & -320 & -320 & -128 & 128 & 320 \\
203 & 29 & -181 & -224 & -67 & 148 & 232 & 116 & -116 & -232 & -148 & 67 & 224 & 181 & -29 & -203 \\
256 & -64 & -304 & -176 & 176 & 304 & 64 & -256 & -256 & 64 & 304 & 176 & -176 & -304 & -64 & 256 \\
392 & -248 & -472 & 56 & 488 & 136 & -424 & -296 & 296 & 424 & -136 & -488 & -56 & 472 & 248 & -392 \\
256 & -256 & -256 & 256 & 256 & -256 & -256 & 256 & 256 & -256 & -256 & 256 & 256 & -256 & -256 & 256 \\
296 & -424 & -136 & 488 & -56 & -472 & 248 & 392 & -392 & 248 & 472 & 56 & -488 & 136 & 424 & -296 \\
176 & -304 & 64 & 256 & -256 & -64 & 304 & -176 & -176 & 304 & -64 & -256 & 256 & 64 & -304 & 176 \\
116 & -232 & 148 & 67 & -224 & 181 & 29 & -203 & 203 & -29 & -181 & 224 & -67 & -148 & 232 & -116 \\
128 & -320 & 320 & -128 & -128 & 320 & -320 & 128 & 128 & -320 & 320 & -128 & -128 & 320 & -320 & 128 \\
56 & -188 & 232 & -203 & 116 & 29 & -139 & 227 & -227 & 139 & -29 & -116 & 203 & -232 & 188 & -56 \\
40 & -120 & 184 & -216 & 216 & -184 & 120 & -40 & -40 & 120 & -184 & 216 & -216 & 184 & -120 & 40 \\
48 & -88 & 168 & -216 & 272 & -304 & 336 & -344 & 344 & -336 & 304 & -272 & 216 & -168 & 88 & -48
\end{bmatrix} \tag{2.34}$$

It has been reported that non-orthogonal transforms can give a high coding performance [19]. The error caused by its non-orthogonality can be controlled at a very low level with proper design of the transform. However, it may not be easy to derive its fast algorithm because of its non-orthogonality. In this thesis, the focus is on the orthogonal order-16 transforms with fast algorithms.

2.4 Simple Integer Transform

To balance the complexity and the compressibility, here we proposed a novel order-16 simple integer transform [20]. It was adopted into the AVS reference software [21]. It is developed from the existing order-8 ICT. Suppose an order-8 ICT with integer kernel $\mathbf{E}_{IC}^{(8)}$ as shown in (2.14), we can extend it into order-16 integer kernel. Here we name them as $\mathbf{E}_{S1}$ and $\mathbf{E}_{S2}$ respectively. They are defined as:

$$\hat{\mathbf{E}}_{S1} = \mathbf{H} \otimes \mathbf{E}_{IC}^{(8)} = \begin{bmatrix} a_0 & a_0 & a_0 & a_0 & a_0 & \cdots & a_0 & a_0 \\ a_0 & -a_0 & a_0 & -a_0 & a_0 & \cdots & a_0 & -a_0 \\ a_1 & a_1 & a_2 & a_2 & a_3 & \cdots & -a_1 & -a_1 \\ a_1 & -a_1 & a_2 & -a_2 & a_3 & \cdots & -a_1 & a_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_6 & -a_6 & -a_5 & a_5 & a_5 & \cdots & -a_6 & a_6 \\ a_4 & a_4 & -a_3 & -a_3 & a_2 & \cdots & -a_4 & -a_4 \\ a_4 & -a_4 & -a_3 & a_3 & a_2 & \cdots & -a_4 & a_4 \end{bmatrix}, \tag{2.35}$$

$$\hat{\mathbf{E}}_{S2} = \mathbf{E}_{IC}^{(8)} \otimes \mathbf{H} = \begin{bmatrix} \mathbf{E}_{IC}^{(8)} & \mathbf{E}_{IC}^{(8)} \\ \mathbf{E}_{IC}^{(8)} & -\mathbf{E}_{IC}^{(8)} \end{bmatrix} = \begin{bmatrix} a_0 & a_0 & a_0 & a_0 & a_0 & \cdots & a_0 & a_0 \\ a_1 & a_2 & a_3 & a_4 & -a_4 & \cdots & -a_2 & -a_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_4 & -a_3 & a_2 & -a_1 & a_1 & \cdots & a_3 & -a_4 \\ a_0 & a_0 & a_0 & a_0 & a_0 & \cdots & -a_0 & -a_0 \\ a_1 & a_2 & a_3 & a_4 & -a_4 & \cdots & a_2 & a_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_4 & -a_3 & a_2 & -a_1 & a_1 & \cdots & -a_3 & a_4 \end{bmatrix}, \tag{2.36}$$

where $\mathbf{H}$ = [1 1; 1 -1] is the order-2 Hadamard Transform Matrix and $\otimes$ is the Kronecker product. We can rearrange $\hat{\mathbf{E}}_{S1}$ and $\hat{\mathbf{E}}_{S2}$ in sequency order to become $\mathbf{E}_{S1}$ and $\mathbf{E}_{S2}$:

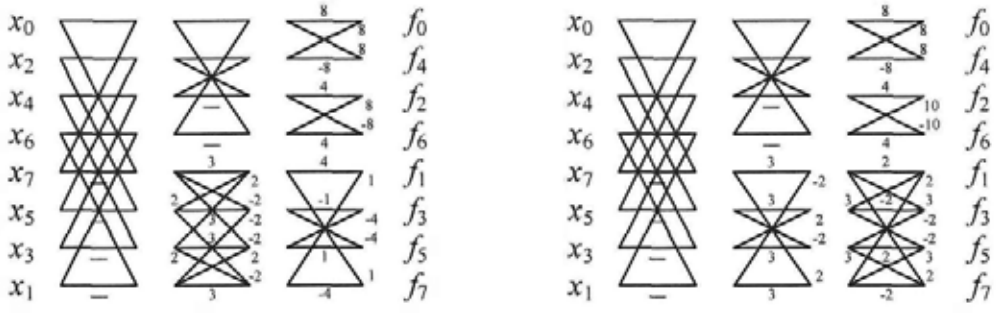
$$\mathbf{E}_{S1}=\begin{bmatrix}
a_0 & a_0 & a_0 & a_0 & a_0 & a_0 & a_0 & a_0 & a_0 & a_0 & \cdots & a_0 \\
a_1 & a_1 & a_2 & a_2 & a_3 & a_3 & a_4 & a_4 & -a_4 & -a_4 & \cdots & -a_1 \\
a_5 & a_5 & a_6 & a_6 & -a_6 & -a_6 & -a_5 & -a_5 & -a_5 & -a_5 & \cdots & a_5 \\
a_2 & a_2 & -a_4 & -a_4 & -a_1 & -a_1 & -a_3 & -a_3 & a_3 & a_3 & \cdots & -a_2 \\
a_0 & a_0 & -a_0 & -a_0 & -a_0 & -a_0 & a_0 & a_0 & a_0 & a_0 & \cdots & a_0 \\
a_3 & a_3 & -a_1 & -a_1 & a_4 & a_4 & a_2 & a_2 & -a_2 & -a_2 & \cdots & -a_3 \\
a_6 & a_6 & -a_5 & -a_5 & a_5 & a_5 & -a_6 & -a_6 & -a_6 & -a_6 & \cdots & a_6 \\
a_4 & a_4 & -a_3 & -a_3 & a_2 & a_2 & -a_1 & -a_1 & a_1 & a_1 & \cdots & -a_5 \\
a_4 & -a_4 & -a_3 & a_3 & a_2 & -a_2 & -a_1 & a_1 & a_1 & -a_1 & \cdots & a_4 \\
a_6 & -a_6 & -a_5 & a_5 & a_5 & -a_5 & -a_6 & a_6 & -a_6 & a_6 & \cdots & -a_6 \\
a_3 & -a_3 & -a_1 & a_1 & a_4 & -a_4 & a_2 & -a_2 & -a_2 & a_2 & \cdots & a_3 \\
a_0 & -a_0 & -a_0 & a_0 & -a_0 & a_0 & a_0 & -a_0 & a_0 & -a_0 & \cdots & -a_0 \\
a_2 & -a_2 & -a_4 & a_4 & -a_1 & a_1 & -a_3 & a_3 & a_3 & -a_3 & \cdots & a_2 \\
a_5 & -a_5 & a_6 & -a_6 & -a_6 & a_6 & -a_5 & a_5 & -a_5 & a_5 & \cdots & -a_5 \\
a_1 & -a_1 & a_2 & -a_2 & a_3 & -a_3 & a_4 & -a_4 & -a_4 & a_4 & \cdots & a_1 \\
a_0 & -a_0 & a_0 & -a_0 & a_0 & -a_0 & a_0 & -a_0 & a_0 & -a_0 & \cdots & -a_0
\end{bmatrix}, \tag{2.37}$$

$$\mathbf{E}_{S2}=\begin{bmatrix}
a_0 & a_0 & a_0 & a_0 & a_0 & a_0 & a_0 & a_0 & a_0 & a_0 & \cdots & a_0 \\
a_0 & a_0 & a_0 & a_0 & a_0 & a_0 & a_0 & a_0 & -a_0 & -a_0 & \cdots & -a_0 \\
a_1 & a_2 & a_3 & a_4 & -a_4 & -a_3 & -a_2 & -a_1 & -a_1 & -a_2 & \cdots & a_1 \\
a_1 & a_2 & a_3 & a_4 & -a_4 & -a_3 & -a_2 & -a_1 & a_1 & a_2 & \cdots & -a_1 \\
a_5 & a_6 & -a_6 & -a_5 & -a_5 & -a_6 & a_6 & a_5 & a_5 & a_6 & \cdots & a_5 \\
a_5 & a_6 & -a_6 & -a_5 & -a_5 & -a_6 & a_6 & a_5 & -a_5 & -a_6 & \cdots & -a_5 \\
a_2 & -a_4 & -a_1 & -a_3 & a_3 & a_1 & a_4 & -a_2 & -a_2 & a_4 & \cdots & a_2 \\
a_2 & -a_4 & -a_1 & -a_3 & a_3 & a_1 & a_4 & -a_2 & a_2 & -a_4 & \cdots & -a_2 \\
a_0 & -a_0 & -a_0 & a_0 & a_0 & -a_0 & -a_0 & a_0 & a_0 & -a_0 & \cdots & a_0 \\
a_0 & -a_0 & -a_0 & a_0 & a_0 & -a_0 & -a_0 & a_0 & -a_0 & a_0 & \cdots & -a_0 \\
a_3 & -a_1 & a_4 & a_2 & -a_2 & -a_4 & a_1 & -a_3 & -a_3 & a_1 & \cdots & a_3 \\
a_3 & -a_1 & a_4 & a_2 & -a_2 & -a_4 & a_1 & -a_3 & a_3 & -a_1 & \cdots & -a_3 \\
a_6 & -a_5 & a_5 & -a_6 & -a_6 & a_5 & -a_5 & a_6 & a_6 & -a_5 & \cdots & a_6 \\
a_6 & -a_5 & a_5 & -a_6 & -a_6 & a_5 & -a_5 & a_6 & -a_6 & a_5 & \cdots & -a_6 \\
a_4 & -a_3 & a_2 & -a_1 & a_1 & -a_2 & a_3 & -a_4 & -a_4 & a_3 & \cdots & a_4 \\
a_4 & -a_3 & a_2 & -a_1 & a_1 & -a_2 & a_3 & -a_4 & a_4 & -a_3 & \cdots & -a_4
\end{bmatrix}. \tag{2.38}$$

Finally, normalize the integer kernels $\mathbf{E}_{S1}$ and $\mathbf{E}_{S2}$ with $\mathbf{K}_{S1}$ and $\mathbf{K}_{S2}$ respectively. Orthogonal order-16 Simple Integer Transforms $\mathbf{T}_{S1}$ and $\mathbf{T}_{S2}$ are formed. Here we name the above two transform matrices to be $\mathbf{T}_{S1}(a_0, a_1 \ldots a_6)$ and $\mathbf{T}_{S2}(a_0, a_1 \ldots a_6)$ respectively. The complexities for these two transforms are the same for the same given set of a_i but $\mathbf{T}_{S1}$ has a higher compressibility than $\mathbf{T}_{S2}$. As a result, we shall demonstrate the experimental result of $\mathbf{T}_{S1}$ only. Two different order-8 ICTs were proposed in H.264/AVC and AVS respectively. For simplicity, the $\mathbf{T}_{S1}$ implemented with the order-8 ICT in H.264/AVC and AVS are denoted as $\mathbf{T}_{S1\text{-}H264}$ and $\mathbf{T}_{S1\text{-}AVS}$

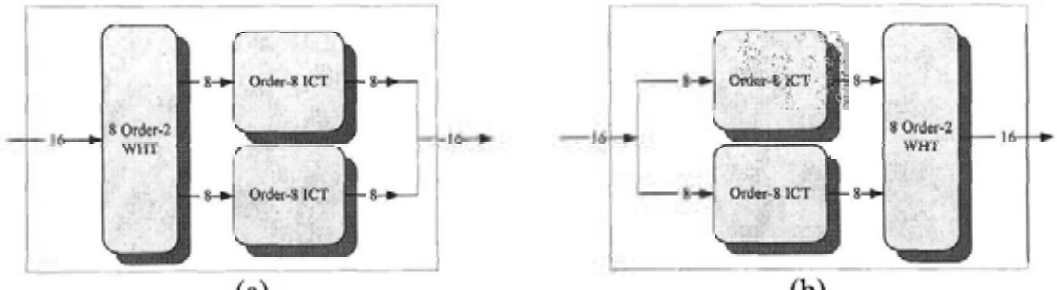
respectively. They are equivalent to $\mathbf{T}_{S1}(8, 12, 10, 6, 3, 8, 4)$ and $\mathbf{T}_{S1}(8, 10, 9, 6, 2, 10, 4)$ respectively. Their integer kernels $\mathbf{E}_{S1}$ can be easily obtained using (2.37).

The fast algorithms for $\mathbf{E}_{S1}$ and $\mathbf{E}_{S2}$ are very simple. Each of them consists of eight order-2 Hadamard Transforms and two order-8 fast ICT. The fast algorithms of these two order-8 ICT are shown in Figure 2-1. The two different Simple Integer Transforms can be derived from these two ICT.



(a) H.264/AVC (b) AVS

Figure 2-1 Fast algorithms of order-8 ICT adopted in (a) H.264/AVC and (b) AVS.



(a) (b)

Figure 2-2 Fast algorithm of proposed order-16 transform (a) $\mathbf{T}_{S1}$ and (b) $\mathbf{T}_{S2}$.

2.5 Hybrid Integer Transform from Dyadic Weighted Walsh Transform

In Simple Integer Transform $\mathbf{T}_{SI}$, there are only 8 different levels in the odd frequency vectors at most. However, there are 16 different levels in DCT. This means that Simple Integer Transform only approximates the DCT with limited accuracy. In order to make the approximation more accurate, the order-8 transform for odd frequencies is replaced by another transform with more levels. In this section, it is replaced by an order-8 Dyadic Weighted Walsh Transform (DWWT) which has a maximum of 8 different levels in each basis vector. As a result, a better approximation and hence a better performance can be achieved.

First of all, let us have a short introduction to DWWT first. For simplicity, here we take an order-4 as an example. Consider a vector of 4 positive elements, $\vec{V}_0 = [a_0, a_1, a_2, a_3]$. We can find its s^{th} dyadic symmetric [22] vector $\vec{V}_s = [b_0, b_1, b_2, b_3]$ by:

$$b_i = a_{j \oplus s} \quad \text{for } i, j, s \in [0,1,2,3] \tag{2.39}$$

and $\oplus$ is the bit-wise "exclusive-or" operation. As a result we can find 4 dyadic symmetric vectors, $\vec{V}_0 \ldots \vec{V}_3$.

Assign +/- sign to each element such that $\vec{V}_i \cdot \vec{V}_j = 0$ if $i \neq j$. This sign assignment is not unique such that more than one orthogonal matrix for the same $\{a_0, a_1, a_2, a_3\}$ can be found. In (2.40), two with different sign assignments are shown.

s	$\vec{V}_s$			
0	a_0	a_1	a_2	a_3
1	a_1	a_0	a_3	a_2
2	a_2	a_3	a_0	a_1
3	a_3	a_2	a_1	a_0

Table 2-2 Example of dyadic symmetric vectors

$$\begin{bmatrix} a_0 & a_1 & a_2 & a_3 \\ a_1 & -a_0 & -a_3 & a_2 \\ a_2 & a_3 & -a_0 & -a_1 \\ a_3 & -a_2 & a_1 & -a_0 \end{bmatrix} \begin{bmatrix} a_0 & a_1 & a_2 & a_3 \\ a_1 & -a_0 & a_3 & -a_2 \\ a_2 & -a_3 & -a_0 & a_1 \\ a_3 & a_2 & -a_1 & -a_0 \end{bmatrix}. \tag{2.40}$$

Denote that the 4×4 matrix formed by the four basis vector $\vec{V}_i$ as $\mathbf{T}_{DWW}^{(4)}$, the Dyadic Weighted Walsh Transform (DWWT). Its basis vectors must be orthogonal for any real $\{a_0, a_1, a_2, a_3\}$. Using the same idea, it can be extended to order-2^N (for $N \geq 2$) transform. One of the order-8 DWWT in its family is:

$$\mathbf{T}_{DWW}^{(8)} = \begin{bmatrix} b_0 & b_1 & b_2 & b_3 & b_4 & b_5 & b_6 & b_7 \\ b_4 & b_5 & b_6 & b_7 & -b_0 & -b_1 & -b_2 & -b_3 \\ b_2 & b_3 & -b_0 & -b_1 & -b_6 & -b_7 & b_4 & b_5 \\ b_6 & b_7 & -b_4 & -b_5 & b_2 & b_3 & -b_0 & -b_1 \\ b_1 & -b_0 & -b_3 & b_2 & b_5 & -b_4 & -b_7 & b_6 \\ b_5 & -b_4 & -b_7 & b_6 & -b_1 & b_0 & b_3 & -b_2 \\ b_3 & -b_2 & b_1 & -b_0 & -b_7 & b_6 & -b_5 & b_4 \\ b_7 & -b_6 & b_5 & -b_4 & b_3 & -b_2 & b_1 & -b_0 \end{bmatrix}. \tag{2.41}$$

Here we name the above order-8 DWWT as $\mathbf{T}_{DWW}$ in later sections for simplicity. It is denoted as $\mathbf{T}_{DWW}(b_0 \ldots b_7)$ when its parameters are given.

Our proposed order-16 Hybrid Integer Transform (HIT), $\mathbf{T}_{HI}^{(16)}$, is a hybrid of two different order-8 transform. One is the above DWWT $\mathbf{T}_{DWW}$ and the other is order-8 ICT $\mathbf{T}_{IC}^{(8)}$. The odd basis vectors are built by $\mathbf{T}_{DWW}$ while the even basis vectors are

built by $\mathbf{T}_{IC}^{(8)}$. $\mathbf{T}_{HI}^{(16)}$ is defined in (2.42).

$$\mathbf{T}_{HI}^{(16)} = \left[\begin{array}{cccccccc:cccc}
a_0 & a_0 & a_0 & a_0 & a_0 & a_0 & a_0 & a_0 & a_0 & a_0 & \cdots & a_0 \\
b_0 & b_1 & b_2 & b_3 & b_4 & b_5 & b_6 & b_7 & -b_7 & -b_6 & \cdots & -b_0 \\
a_1 & a_2 & a_3 & a_4 & -a_4 & -a_3 & -a_2 & -a_1 & -a_1 & -a_2 & \cdots & a_1 \\
b_2 & b_5 & b_6 & b_7 & -b_0 & -b_1 & -b_2 & -b_3 & b_3 & b_2 & \cdots & -b_4 \\
a_5 & a_6 & -a_6 & -a_5 & -a_5 & -a_6 & a_6 & a_5 & a_5 & a_6 & \cdots & a_5 \\
b_2 & b_3 & -b_0 & -b_1 & b_6 & b_7 & b_4 & b_5 & -b_5 & -b_4 & \cdots & -b_4 \\
a_2 & -a_4 & -a_1 & -a_3 & a_3 & a_1 & a_4 & -a_2 & -a_2 & a_4 & \cdots & a_2 \\
b_6 & b_7 & -b_4 & -b_5 & b_2 & b_3 & -b_0 & -b_1 & b_1 & b_0 & \cdots & -b_6 \\
\hdashline
a_0 & -a_0 & -a_0 & a_0 & a_0 & -a_0 & -a_0 & a_0 & a_0 & -a_0 & \cdots & a_0 \\
b_1 & -b_0 & -b_3 & b_2 & b_5 & -b_4 & -b_7 & b_6 & -b_6 & b_7 & \cdots & -b_1 \\
a_3 & -a_1 & a_4 & a_2 & -a_2 & -a_4 & a_1 & -a_3 & -a_3 & a_1 & \cdots & a_3 \\
b_5 & -b_4 & -b_7 & b_6 & -b_1 & b_0 & b_3 & -b_2 & b_2 & -b_3 & \cdots & -b_5 \\
a_6 & -a_5 & a_5 & -a_6 & -a_6 & a_5 & -a_5 & a_6 & a_6 & -a_5 & \cdots & a_6 \\
b_3 & -b_2 & b_1 & -b_0 & -b_7 & b_6 & -b_5 & b_4 & -b_4 & b_5 & \cdots & -b_3 \\
a_4 & -a_3 & a_2 & -a_1 & a_1 & -a_2 & a_3 & -a_4 & -a_4 & a_3 & \cdots & a_4 \\
b_7 & -b_6 & b_5 & -b_4 & b_3 & -b_2 & b_1 & -b_0 & b_0 & -b_1 & \cdots & -b_7
\end{array}\right] \tag{2.42}$$

In matrix representation, it is represented as:

$$\mathbf{T}_{HI}^{(16)} = \frac{1}{\sqrt{2}}\begin{bmatrix} \mathbf{T}_{IC}^{(8)} & \mathbf{0} \\ \mathbf{0} & \mathbf{T}_{DWW}^{(8)} \end{bmatrix}\begin{bmatrix} \boldsymbol{I}_8 & \widetilde{\boldsymbol{I}}_8 \\ I_8 & -\widetilde{I}_8 \end{bmatrix} \tag{2.43}$$

HIT can be expressed as:

$$\mathbf{T}_{HI}^{(16)} = \mathbf{K}_{HI}^{(16)}\mathbf{E}_{HI}^{(16)} \tag{2.44}$$

such that $\mathbf{K}_{HI}^{(16)}$ is a diagonal matrix to unify the basis vectors of the integer kernel $\mathbf{E}_{HI}^{(16)}$ and hence $\mathbf{T}_{HI}^{(16)}$ is orthogonal. $\mathbf{E}_{HI}^{(16)}$ is built by $\mathbf{E}_{IC}^{(8)}$ and $\mathbf{E}_{DWW}$, which are the integer kernels of $\mathbf{T}_{IC}^{(8)}$ and $\mathbf{T}_{DWW}^{(8)}$ respectively. For $\mathbf{E}_{IC}^{(8)}$, here we can simply choose the integer kernel of $\mathrm{ICT}_8(8, 10, 9, 6, 2, 8, 4)$ which is adopted in AVS. For $\mathbf{E}_{DWW}$, because of its property, there are infinite many solutions. Here we search for different $\mathbf{E}_{DWW}$ which produce $\mathbf{T}_{HI}^{(16)}$ with high coding gain G_{TC} (see Section 2.5) with basis norm similar to that of $\mathrm{ICT}_8(8, 10, 9, 6, 2, 8, 4)$. These sets of $\mathbf{E}_{DWW}$ are

shown in the following table. Besides compressibility, existence of fast algorithm is also an important concern. The fast algorithm for $\mathbf{E}_{HI}^{(16)}$ is achieved as shown in Figure 2-3.

ID	Set of $\mathbf{E}_{DWW}$									G_{TC} (ρ=0.9)
	Norm	b_0	b_1	b_2	b_3	b_4	b_5	b_6	b_7	
$\mathbf{E}_{DWW1}$	574	11	11	11	9	8	7	4	1	4.501
$\mathbf{E}_{DWW2}$	570	11	11	11	9	8	6	5	1	4.497
$\mathbf{E}_{DWW3}$ **(MICT)**	561	11	11	11	9	8	6	4	1	4.508
$\mathbf{E}_{DWW4}$	546	11	11	11	9	7	6	4	1	4.498
$\mathbf{E}_{DWW5}$	544	11	11	11	8	8	6	4	1	4.493
$\mathbf{E}_{DWW6}$	540	11	11	10	9	8	6	4	1	4.987

Table 2-3 Examples of $\mathbf{E}_{DWW}$ with high coding gain G_{TC}.

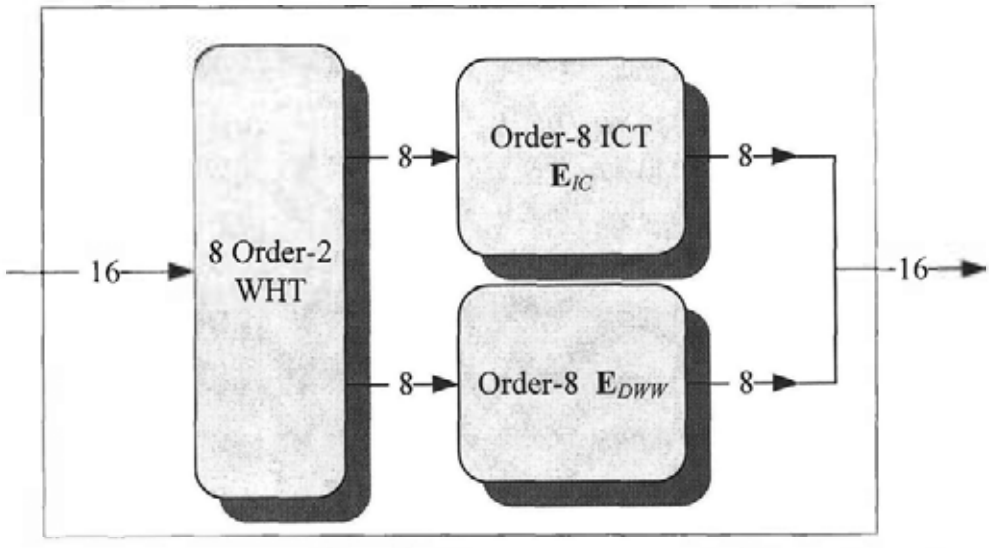


Figure 2-3 The general fast algorithm of proposed order-16 transform $\mathbf{E}_{III}$.

It is not easy to obtain a generalized fast algorithm for $\mathbf{E}_{DWW}$ but for a specific $\mathbf{E}_{DWW}$. In [19], Dong pointed out that orthogonal matrix can be decomposed into product of simpler sparse matrices if its norm is not prime. Let the norm of $\mathbf{E}_{DWW}$ be D. If $\mathbf{E}_{DWW}$ can be factorized such that $\mathbf{E}_{DWW} = \mathbf{M}_1\mathbf{M}_2\mathbf{M}_3\ldots\mathbf{M}_{k-1}$, D can also be factorized

into $d_1 d_2 \ldots d_k = D$ where d_i is the norm of $\mathbf{M}_i$. So if the factors of D are found, the factors of $\mathbf{E}_{DWW}$ can be easily found. However, some D are prime numbers or have large prime factors. It is difficult to factorize these $\mathbf{E}_{DWW}$. For these $\mathbf{E}_{DWW}$, they can be decomposed into simpler linear sum first. This can simplify the factors of $\mathbf{E}_{DWW}$ and provide a simpler fast algorithm for $\mathbf{E}_{DWW}$. An example the decomposition of $\mathbf{E}_{DWW1}$ is shown in (2.45). It is decomposed into the sum of two $\mathbf{E}_{DWW}$, $\mathbf{E}_{DWW}(11, 11, 11, 9, 8, 5, 4, 1)$ and $\mathbf{E}_{DWW}(0, 0, 0, 0, 0, 2, 0, 0)$. The first one can be decomposed into 4 sparse matrices.

$$
\begin{aligned}
&\mathbf{E}_{DWW1} \\
&= \mathbf{E}_{DWW}(11,11,11,9,8,7,4,1) \\
&= \mathbf{E}_{DWW}(11,11,11,9,8,5,4,1) + \mathbf{E}_{DWW}(0,0,0,0,0,2,0,0) \\
&= \begin{bmatrix}
-1 & & 2 & 1 & & & 2 & -1 \\
-1 & & -2 & & & 2 & 1 & -1 \\
 & -1 & 1 & & -1 & 2 & & 2 \\
2 & 1 & & 2 & 1 & 1 & & \\
 & 1 & -1 & & & -1 & 2 & 2 \\
2 & & & -1 & -2 & & 1 & -1 \\
 & 2 & 1 & -2 & 1 & 1 & & \\
-1 & 2 & & 1 & -2 & & -1 &
\end{bmatrix}
\begin{bmatrix}
1 & & & & -1 & 1 & -1 & 1 \\
 & 1 & & & 1 & -1 & -1 & 1 \\
 & & 1 & & 1 & 1 & 1 & 1 \\
 & & & 1 & 1 & -1 & -1 & -1 \\
1 & -1 & -1 & 1 & & & & 1 \\
1 & -1 & 1 & & -1 & & & -1 \\
1 & 1 & -1 & & & 1 & & -1 \\
1 & 1 & 1 & & & & -1 & 1
\end{bmatrix} \\
&\times \begin{bmatrix}
1 & -1 & & 1 & & & 1 & 1 \\
 & 1 & -1 & 1 & & & -1 & 1 \\
 & 1 & & -1 & -1 & & 1 & 1 \\
 & 1 & & 1 & & 1 & & -1 \\
1 & 1 & 1 & & 1 & -1 & & \\
1 & & -1 & -1 & 1 & 1 & & \\
1 & & 1 & & -1 & 1 & -1 & \\
1 & & -1 & & -1 & -1 & & -1
\end{bmatrix}
\begin{bmatrix}
1 & & 1 & & & & & \\
 & 1 & & 1 & & & & \\
-1 & & 1 & & & & & \\
 & -1 & & 1 & & & & \\
 & & & & 1 & & 1 & \\
 & & & & & 1 & & 1 \\
 & & & & -1 & & 1 & \\
 & & & & & -1 & & 1
\end{bmatrix} \\
&+ \begin{bmatrix}
 & & & & 2 & & & \\
 & 2 & & & & & & \\
 & & & & & & 2 & \\
 & & -2 & & & & & \\
 & & & 2 & & & & \\
 & & & & -2 & & & \\
 & & & & & -2 & & \\
2 & & & & & & &
\end{bmatrix}
\end{aligned}
\qquad (2.45)
$$

The factorizations of the $\mathbf{E}_{DWW1}$, $\mathbf{E}_{DWW2}$ and $\mathbf{E}_{DWW3}$ are shown in the appendix. With the factorization, the fast algorithm of $\mathbf{E}_{DWW}$, and hence that of $\mathbf{E}_{HI}^{(16)}$, can be obtained.

2.6 LLM Integer Cosine Transform

2.6.1 Relaxed GCT

The proposed order-16 transforms in previous sections are extended from order-8 transforms. The waveforms of these transform is slightly different from those in the DCT. This lowers the coding performance. The order-16 ICT proposed in [14] is very close to the DCT. However, it is not easy to derive its fast algorithm and no fast algorithm has been proposed so far. As an alternative, here we proposed a novel method to derive higher order ICT based on an existing fast DCT algorithm.

Recall that GCT in (2.9) can be decomposed into two parts, $\mathbf{P}_{GC}^{(N)}$ and $\mathbf{Q}_{GC}^{(N)}$. Here we proposed to relax $\mathbf{P}_{GC}^{(N)}$ to $\mathbf{P}_{RGC}^{(N)}$ such that:

$$\mathbf{P}_{RGC}^{(N)} = \begin{bmatrix} a_1 & a_2 & a_3 & a_4 & a_5 & \cdots & a_{N-2} & a_{N-1} \\ b_2 & \cdots & & & & & \cdots & -b_{N-2} \\ c_3 & \cdots & & & & & \cdots & c_{N-3} \\ \vdots & & & & & & & \vdots \\ c_{N-3} & \cdots & & & & & \cdots & -c_3 \\ b_{N-2} & \cdots & & & & & \cdots & b_2 \\ a_{N-1} & -a_{N-2} & \cdots & -a_5 & a_4 & -a_3 & a_2 & -a_1 \end{bmatrix}. \tag{2.46}$$

The N-element matrix $\mathbf{P}_{GC}^{(N)}$ is relaxed to an $(N^2/2)$-element matrix $\mathbf{P}_{RGC}^{(N)}$. Here, Relaxed General Cosine Transform (RGCT) is defined by replacing $\mathbf{P}_{GC}^{(N)}$ in (2.9) with $\mathbf{P}_{RGC}^{(N)}$:

$$\mathbf{T}_{RGC}^{(2N)} = \begin{bmatrix} \frac{1}{\sqrt{2}}\mathbf{T}_{RGC}^{(N)} & \mathbf{0}_N \\ \mathbf{0}_N & \mathbf{P}_{RGC}^{(N)} \end{bmatrix} \begin{bmatrix} \mathbf{I}_N & \tilde{\mathbf{I}}_N \\ \mathbf{I}_N & -\tilde{\mathbf{I}}_N \end{bmatrix} \text{ for } N \geq 2. \tag{2.47}$$

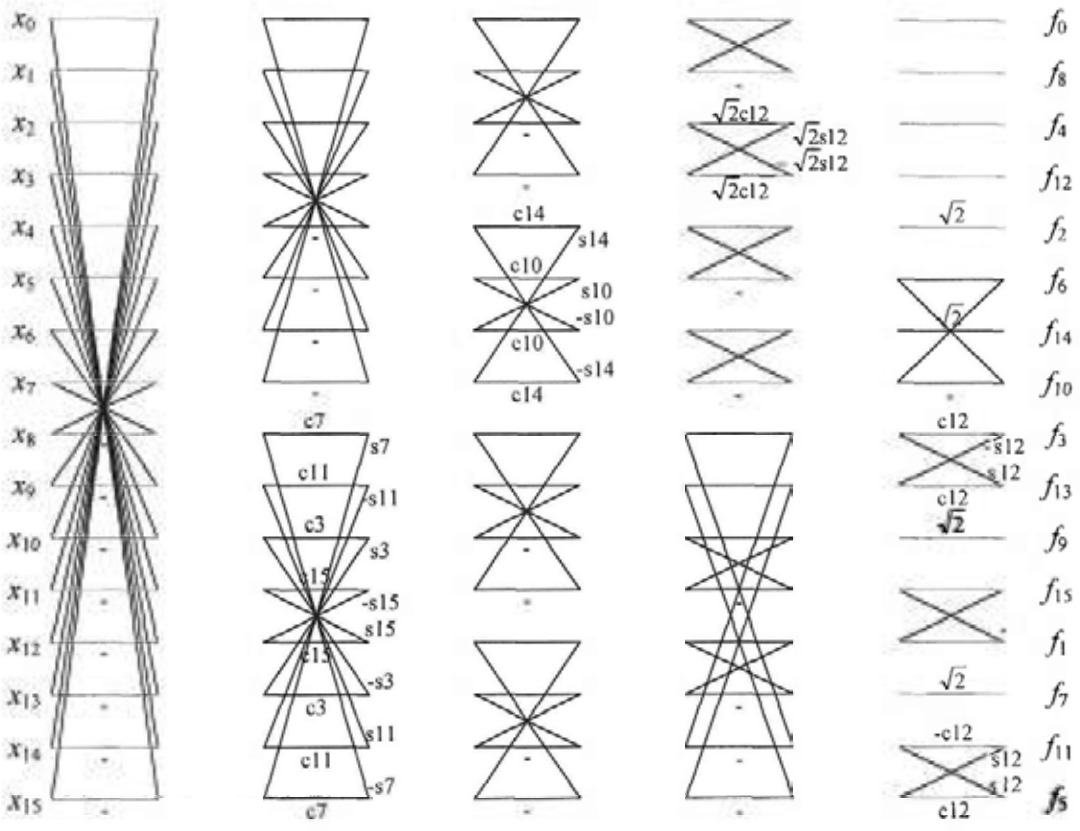
Take $N = 8$ as an example, $\mathbf{P}_{RGC}^{(8)}$ is

$$\mathbf{P}_{RGC}^{(8)} = \begin{bmatrix} a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7 & a_8 \\ b_2 & b_5 & b_8 & -b_6 & -b_3 & -b_1 & -b_4 & -b_7 \\ c_3 & c_8 & -c_4 & -c_2 & -c_7 & c_5 & c_1 & c_6 \\ d_4 & -d_6 & -d_2 & d_8 & d_1 & d_7 & -d_3 & -d_5 \\ d_5 & -d_3 & -d_7 & d_1 & -d_8 & -d_2 & d_6 & d_4 \\ c_6 & -c_1 & c_5 & c_7 & -c_2 & c_4 & c_8 & -c_3 \\ b_7 & -b_4 & b_1 & -b_3 & b_6 & b_8 & -b_5 & b_2 \\ a_8 & -a_7 & a_6 & -a_5 & a_4 & -a_3 & a_2 & -a_1 \end{bmatrix}. \quad (2.48)$$

The next step is to find the values in $\mathbf{P}_{RGC}^{(8)}$. This can be achieved by existing fast DCT algorithms such as the LLM fast DCT algorithm proposed by Loeffler et al. [26] and the CSF fast DCT algorithm proposed by Chen et al. [23].

2.6.2 The LLM Fast DCT

LLM fast DCT algorithm [26], which was proposed by Loeffler et al., requires only a few multiplications. It requires only 31 multiplications and 81 additions for order-16 1-D DCT. It is also presented in a nice butterfly structure shown in Figure 2-4. It is divided into 5 stages from the input signal $x_{0...15}$ to the output coefficients $f_{0...15}$. Multiplications with the irrational constants shown in the figure are required. Although only order-8 and order-16 LLM Fast DCT are proposed in [26], it is not difficult to derived higher order fast algorithm with similar structure. As a result, we use it in our proposed algorithm of deriving higher order ICT. This will be shown in later section.



$$c_k = \cos\left(\frac{k\pi}{32}\right), \quad s_k = \sin\left(\frac{k\pi}{32}\right).$$

Figure 2-4 The LLM fast DCT algorithm.

2.6.3 The Proposed LLMICT

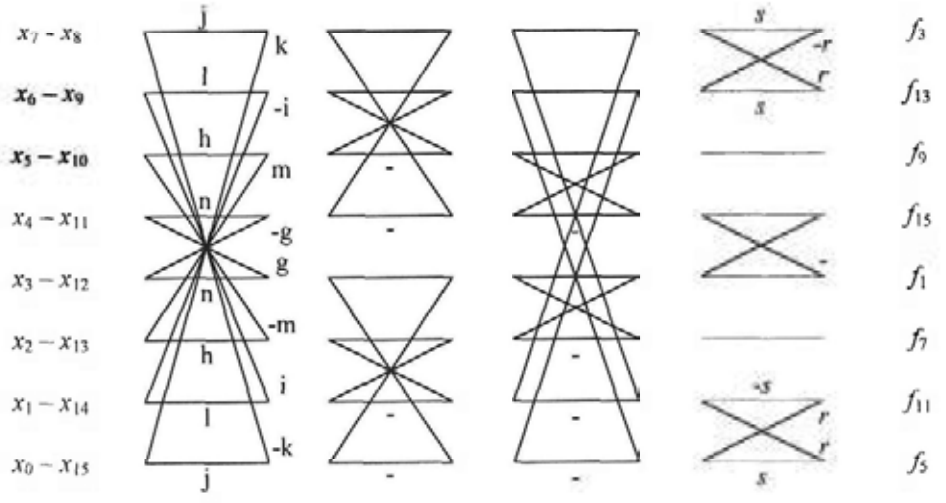


Figure 2-5 The generalized odd part of LLM algorithm.

Since we are going to find only the odd part, $\mathbf{P}_{RGC}^{(8)}$, we focus in the odd part of the LLM algorithm. Its generalized version is shown in Figure 2-5. The irrational constants are replaced by variables $g...n$, r and s. It is very intuitive to take these constant as the integer approximation of the irrational constants. For example $h = 61 \approx 61.2442 \approx 2^6 \times \cos\left(\frac{3\pi}{32}\right)$. However, the orthogonality will be destroyed by doing so. Instead, when the fast algorithm is expanded into matrix form, the matrix of the odd part becomes:

$$\mathbf{P}_{LLM}^{(8)} = \begin{bmatrix} j+k & i+l & h+m & g+n & g-n & h-m & i-l & j-k \\ rj+sk & ri-sl & sh-rm & -(sg+rn) & -(rg-sn) & -(rh+sm) & -(si+rl) & -(rk-sj) \\ sj+rk & rl-si & -(rh-sm) & -(rg+sn) & -(sg-rn) & sh+rm & ri+sl & rj-sk \\ j & -l & -h & n & g & m & -i & -k \\ k & -i & -m & g & -n & -h & l & j \\ rj-sk & -(ri+sl) & sh+rm & sg-rn & -(rg+sn) & rh-sm & rl-si & -(sj+rk) \\ rk-sj & -(si+rl) & rh+sm & rg-sn & -(sg+rn) & sh-rm & -(ri-sl) & rj+sk \\ j-k & -(i-l) & h-m & -(g-n) & g+n & -(h+m) & i+l & -(j+k) \end{bmatrix}, \quad (2.49)$$

Compare (2.49) with (2.48), we can find that they have the same structure and hence $\mathbf{P}_{LLM}^{(8)}$ is a possible solution to $\mathbf{P}_{RGC}^{(8)}$. Here, we propose an orthogonal integer transform $\mathbf{T}_{LLM}^{(N)}$, called LLM Integer Cosine Transform (LLMICT) by replacing $\mathbf{P}_{RGC}^{(N)}$ in (2.47) with $\mathbf{P}_{LLM}^{(N)}$:

$$\mathbf{T}_{LLM}^{(2N)} = \begin{bmatrix} \frac{1}{\sqrt{2}}\mathbf{T}_{LLM}^{(N)} & \mathbf{0}_N \\ \mathbf{0}_N & \mathbf{P}_{LLM}^{(N)} \end{bmatrix} \begin{bmatrix} \mathbf{I}_N & \widetilde{\mathbf{I}}_N \\ \mathbf{I}_N & -\widetilde{\mathbf{I}}_N \end{bmatrix}, \quad (2.50)$$

For ease of implementation, $\mathbf{T}_{LLM}^{(N)}$ can be replaced by $\mathbf{T}_{IC}^{(N)}$ when $N \leq 8$. Lastly, to ensure $\mathbf{P}_{LLM}^{(8)}$ orthogonal, the criteria for orthogonality (2.51), (2.52) and (2.53) have to be satisfied for arbitrary r and s:

$$g^2 + n^2 = j^2 + k^2, \tag{2.51}$$

$$h^2 + m^2 = i^2 + l^2, \text{ and} \tag{2.52}$$

$$j^2 + k^2 + g^2 + n^2 = h^2 + m^2 + i^2 + l^2. \tag{2.53}$$

The solution to the above equations is:

$$g^2 + n^2 = h^2 + m^2 = i^2 + l^2 = j^2 + k^2 = \alpha \tag{2.54}$$

for some positive integer α. To make the basis vectors similar to the DCT ones, another constraint is added:

$$g \geq h \geq i \geq j \geq k \geq l \geq m \geq n. \tag{2.55}$$

There are infinite sets of solutions to these two constraints (2.54) and (2.55). It is very interesting that some of these solutions can be found recursively without exhaustive search. If given a solution (g, h, i, j, k, l, m, n) to (2.54) and (2.55), (j+k, i+l, h+m, g+n, g-n, h-m, i-l, j-k) must be a solution, too. This is because:

$$(j+k)^2 + (j-k)^2 = 2(j^2 + k^2) = 2\alpha \tag{2.56}$$

which is also a positive integer. This means that the (j+k, j-k) pair satisfies (2.54). Other pairs, (i+l, i-l), (h+m, h-m) and (g+n, g-n), also have the same property such that (j+k, i+l, h+m, g+n, g-n, h-m, i-l, j-k) is a new set of solution.

Here some examples of (g...n) are shown in Table 2-4 and Table 2-5. (g...n) can be represented in 6 bits in Table 2-4 while they can be represented in 5 bits in Table 2-5.

They are sorted according to their coding gain in descending order in each table. They are named as A1~A7 and B1~B7.

ID	g	h	i	j	k	l	m	n
A1	61	59	53	49	37	31	17	7
A2	46	45	42	35	30	19	10	3
A3	43	42	38	34	27	21	11	6
A4	59	58	53	46	37	26	11	2
A5	62	59	53	46	43	34	22	11
A6	51	48	44	37	36	27	19	8
A7	49	47	44	41	28	23	16	8

Table 2-4 Example solutions for $g...n$ which can be represented in 6 bits.

ID	g	h	i	j	k	l	m	n
B1	18	17	15	15	10	10	6	1
B2	29	26	22	22	19	19	13	2
B3	20	19	16	16	13	13	8	5
B4	26	25	23	23	14	14	10	7
B5	30	27	22	22	21	21	14	5
B6	25	23	19	19	17	17	11	5
B7	29	27	25	25	15	15	11	3

Table 2-5 Example solutions for $g...n$ which can be represented in 5 bits.

The examples in Table 2-5 can be implemented with fewer operations. However, the coding performance will be slightly lowered. This is a trade-off between complexity and performance.

Although r and s can be arbitrary, it is better to choose $r : s \approx \sin\left(\frac{12\pi}{32}\right) : \cos\left(\frac{12\pi}{32}\right) \approx 0.4142$. (r, s) can be taken as (1, 2) or (2, 5) for simplicity.

Orthogonal $\mathbf{P}_{LLM}^{(8)}$ is found. Recall (2.50) and replace $\mathbf{T}_{LLM}^{(8)}$ with $\mathrm{ICT}_8(8, 10, 9, 6, 2, 10, 4)$. Finally, in order to normalize the dynamic ranges of different coefficients,

bit shifts are added to the last stage. Taking A1 and B1 as the example, their integer transform kernels are shown in (2.57) and (2.58). Their 1-D fast transforms are shown in Figure 2-6 and Figure 2-7 respectively.

$$\mathbf{E}_{LLM-A1} = \begin{bmatrix}
256 & 256 & 256 & 256 & 256 & 256 & 256 & 256 & 256 & 256 & 256 & 256 & 256 & 256 & 256 & 256 \\
344 & 336 & 304 & 272 & 216 & 168 & 88 & 48 & -48 & -88 & -168 & -216 & -272 & -304 & -336 & -344 \\
320 & 288 & 192 & 64 & -64 & -192 & -288 & -320 & -320 & -288 & -192 & -64 & 64 & 192 & 288 & 320 \\
319 & 203 & 33 & -157 & -291 & -329 & -261 & -87 & 87 & 261 & 329 & 291 & 157 & -33 & -203 & -319 \\
320 & 128 & -128 & -320 & -320 & -128 & 128 & 320 & 320 & 128 & -128 & -320 & -320 & -128 & 128 & 320 \\
283 & 49 & -261 & -319 & -87 & 203 & 327 & 171 & -171 & -327 & -203 & 87 & 319 & 261 & -49 & -283 \\
288 & -64 & -320 & -192 & 192 & 320 & 64 & -288 & -288 & 64 & 320 & 192 & -192 & -320 & -64 & 288 \\
196 & -124 & -236 & 28 & 244 & 68 & -212 & -148 & 148 & 212 & -68 & -244 & -28 & 236 & 124 & -196 \\
256 & -256 & -256 & 256 & 256 & -256 & -256 & 256 & 256 & -256 & -256 & 256 & 256 & -256 & -256 & 256 \\
148 & -212 & -68 & 244 & -28 & -236 & 124 & 196 & -196 & -124 & 236 & 28 & -244 & 68 & 212 & -148 \\
192 & -320 & 64 & 288 & -288 & -64 & 320 & -192 & -192 & 320 & -64 & -288 & 288 & 64 & -320 & 192 \\
171 & -327 & 203 & 87 & -319 & 261 & 49 & -283 & 283 & -49 & -261 & 319 & -87 & -203 & 327 & -171 \\
128 & -320 & 320 & -128 & -128 & 320 & -320 & 128 & 128 & -320 & 320 & -128 & -128 & 320 & -320 & 128 \\
87 & -261 & 329 & -291 & 157 & 33 & -203 & 319 & -319 & 203 & -33 & -157 & 291 & -329 & 261 & -87 \\
64 & -192 & 288 & -320 & 320 & -288 & 192 & -64 & -64 & 192 & -288 & 320 & -320 & 288 & -192 & 64 \\
48 & -88 & 168 & -216 & 272 & -304 & 336 & -344 & 344 & -336 & 304 & -272 & 216 & -168 & 88 & -48
\end{bmatrix} \tag{2.57}$$

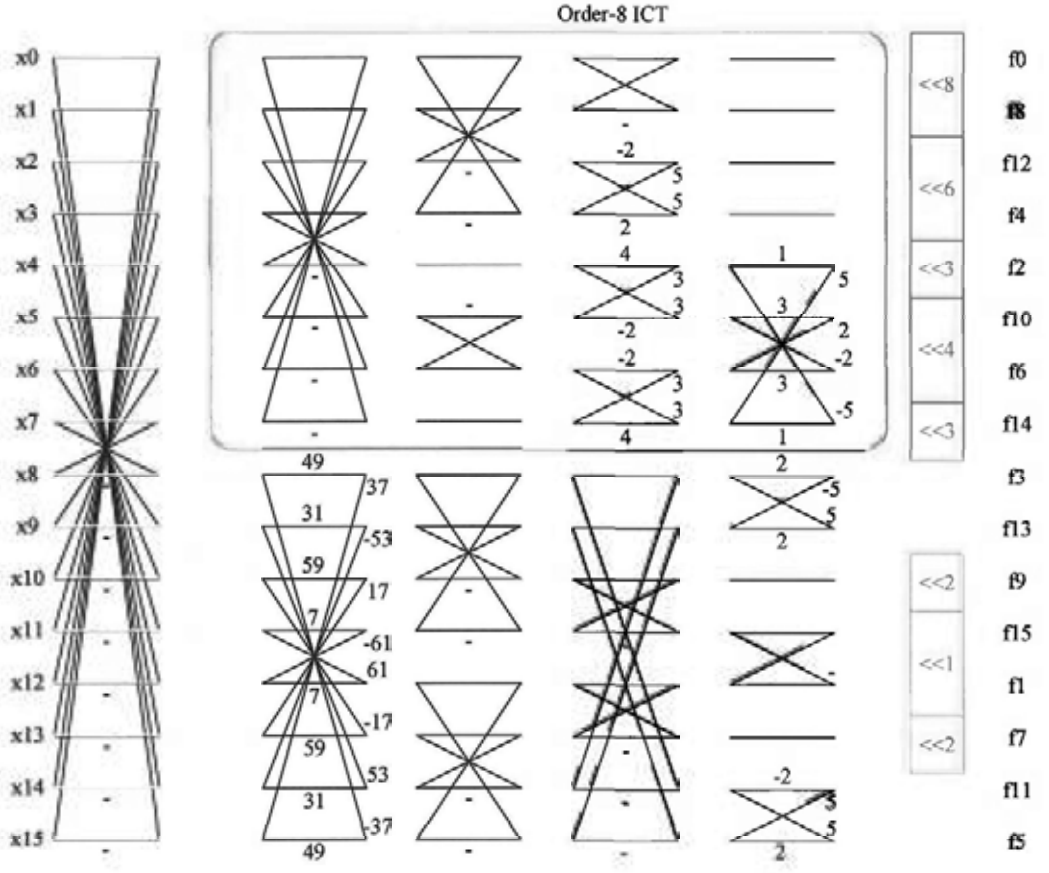


Figure 2-6 Fast 1-D Forward Transform for LLMICT-A1.

$$\mathbf{E}_{LLMICT-B1} = \begin{bmatrix}
32 & 32 & 32 & 32 & 32 & 32 & 32 & 32 & 32 & 32 & 32 & 32 & 32 & 32 & 32 & 32 \\
25 & 25 & 23 & 19 & 17 & 11 & 5 & 5 & -5 & -5 & -11 & -17 & -19 & -23 & -25 & -25 \\
40 & 36 & 24 & 8 & -8 & -24 & -36 & -40 & -40 & -36 & -24 & -8 & 8 & 24 & 36 & 40 \\
40 & 20 & 5 & -20 & -35 & -40 & -35 & -5 & 5 & 35 & 40 & 35 & 20 & -5 & -20 & -40 \\
40 & 16 & -16 & -40 & -40 & -16 & 16 & 40 & 40 & 16 & -16 & -40 & -40 & -16 & 16 & 40 \\
35 & 5 & -28 & -37 & -16 & 29 & 40 & 20 & -20 & -40 & -29 & 16 & 37 & 28 & -5 & -35 \\
36 & -8 & -40 & -24 & 24 & 40 & 8 & -36 & -36 & 8 & 40 & 24 & -24 & -40 & -8 & 36 \\
30 & -20 & -34 & 2 & 36 & 12 & -30 & -20 & 20 & 30 & -12 & -36 & -2 & 34 & 20 & -30 \\
32 & -32 & -32 & 32 & 32 & -32 & -32 & 32 & 32 & -32 & -32 & 32 & 32 & -32 & -32 & 32 \\
20 & -30 & -12 & 36 & -2 & -34 & 20 & 30 & -30 & -20 & 34 & 2 & -36 & 12 & 30 & -20 \\
24 & -40 & 8 & 36 & -36 & -8 & 40 & -24 & -24 & 40 & -8 & -36 & 16 & 8 & -40 & 24 \\
20 & -40 & 29 & 16 & -37 & 28 & 5 & -35 & 35 & -5 & -28 & 37 & -16 & -23 & 40 & -20 \\
16 & -40 & 40 & -16 & -16 & 40 & -40 & 16 & 16 & -40 & 40 & -16 & 35 & 40 & -40 & 16 \\
5 & -35 & 40 & -35 & 20 & 5 & -20 & 40 & -40 & 20 & -5 & -20 & 35 & -40 & 35 & -5 \\
8 & -24 & 36 & -40 & -40 & -36 & 24 & -8 & -8 & 24 & -36 & 40 & -40 & 36 & -24 & 8 \\
5 & -5 & 11 & -17 & 19 & -23 & 25 & -25 & 25 & -25 & 23 & -19 & 17 & 11 & 5 & -5
\end{bmatrix} \tag{2.58}$$

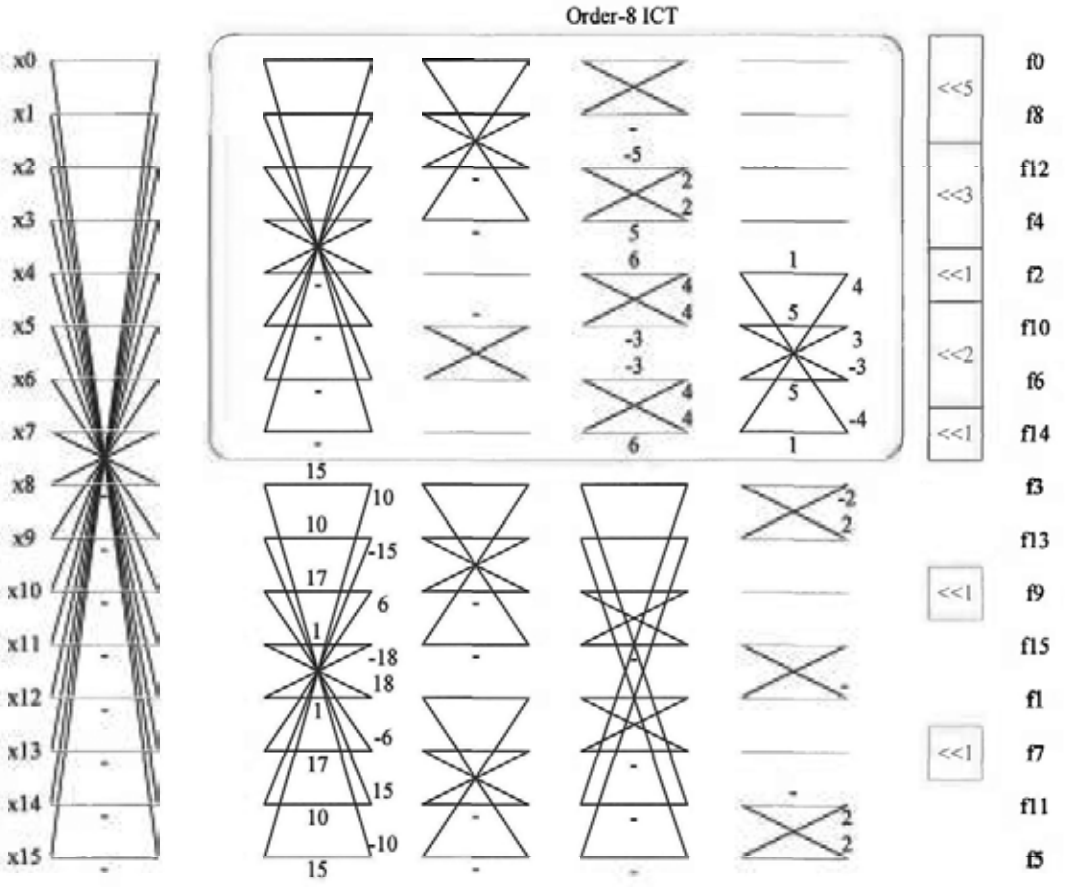


Figure 2-7 Fast 1-D Forward Transform of LLMICT-B1.

One may wonder the relationships among the ICT proposed in [1] and [14] (i.e. $T_{IC}^{(N)}$) and our proposed LLMICT. Their relationship is illustrated in Figure 2-8. In [1] and [14], the DCT is generalized to $\mathbf{T}_{GC}$ and then the ICT, $\mathbf{T}_{IC}$ is found as a subset in $\mathbf{T}_{GC}$. Here, we further generalize $\mathbf{T}_{GC}$ to $\mathbf{T}_{RGCT}$ and a subset, $\mathbf{T}_{LLM}$, is found in $\mathbf{T}_{RGCT}$. If the members in $\mathbf{T}_{LLM}$ have integer kernels, they are our proposed LLMICT or denoted as $\mathbf{T}_{ILLM}$ in Figure 2-8. The approach to derive orthogonal transforms here is different from that in [1] and [14]. However, the fast algorithm of the ICT in [14] (order-16 ICT) may not be found easily.

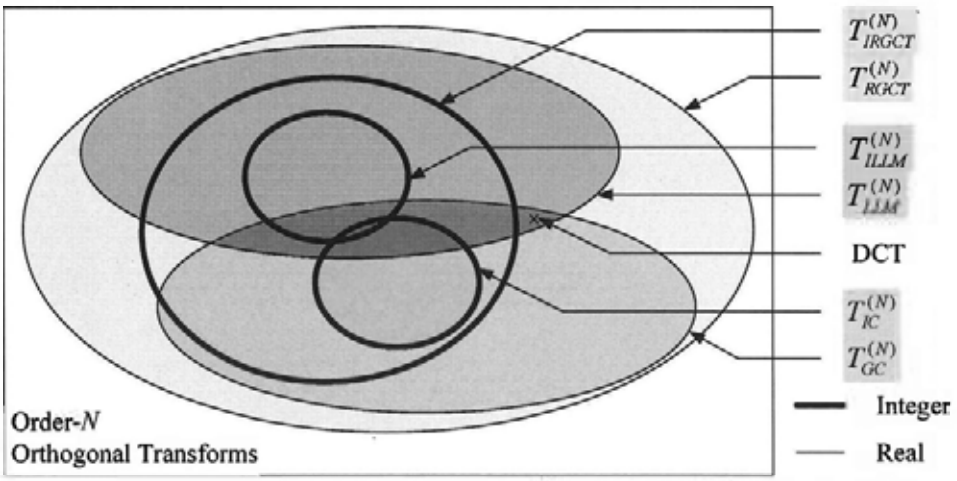


Figure 2-8 The relationship among $T_{IC}^{(N)}$, $T_{RGC}^{(N)}$, $T_{LLM}^{(N)}$ and the DCT.

Despite of the order-16 ICT proposed in [14], it is not easy to derive its fast algorithm. Instead, our proposed algorithm is based on a well-structured fast algorithm with integer multiplication only. The proposed transform has a recursive structure such that higher order transform, such as order-32, can be easily derived. Since the LLM fast DCT is defined for order-2^N only, the proposed LLMICT is also defined for order-2^N only. It is not difficult to derive the LLM algorithm for higher order fast DCT. As a result, $P_{LLM}^{(N)}$ and hence $T_{LLM}^{(N)}$ can be easily found for $N > 16$.

2.6.4 Order-32 LLMICT

Using the same idea, order-32 LLMICT can be found. Its fast algorithm is shown in Figure 2-9. Here, the order-16 LLMICT for the even part is using our proposed LLMICT-A1. In the figure, a_1.. a_{22} are predefined constants. The criteria for orthogonality are:

$$\begin{aligned} &a_1^2 + a_{16}^2 = a_2^2 + a_{15}^2 = a_3^2 + a_{14}^2 = a_4^2 + a_{13}^2 \\ &= a_5^2 + a_{12}^2 = a_6^2 + a_{11}^2 = a_7^2 + a_{10}^2 = a_8^2 + a_9^2 = \alpha \end{aligned} \quad \text{for some positive } \alpha, \tag{2.59}$$

$$a_{17}^2 + a_{20}^2 = a_{18}^2 + a_{19}^2 = \beta \quad \text{for some positive } \beta. \tag{2.60}$$

Three more constraints are added to make the waveforms similar to these of DCT:

$$\begin{aligned} &a_1 \geq a_2 \geq a_3 \geq a_4 \geq a_5 \geq a_6 \geq a_7 \geq a_8 \\ &\geq a_9 \geq a_{10} \geq a_{11} \geq a_{12} \geq a_{13} \geq a_{14} \geq a_{15} \geq a_{16} \end{aligned} \tag{2.61}$$

$$a_{17} \geq a_{18} \geq a_{19} \geq a_{20}, \tag{2.62}$$

$$a_{21} : a_{22} \approx \sin\left(\tfrac{24\pi}{64}\right) : \cos\left(\tfrac{24\pi}{64}\right) \approx 2.4142. \tag{2.63}$$

For simplicity, (a_{21}, a_{22}) are taken as (2, 1). Some solution sets for (2.59), (2.60), (2.61) and (2.62) are shown in Table 2-6, Table 2-7 and Table 2-8.

Set	a_{17}	a_{18}	a_{19}	a_{20}
1	8	7	4	1
2	9	7	6	2
3	11	10	5	2
4	11	9	7	3
5	12	9	8	1

Table 2-6 Example solutions for order-32 LLMICT which satisfy (2.60) and (2.62).

Set	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}	a_{16}
A1	253	251	243	237	233	219	197	181	177	159	127	99	89	71	33	9
A2	179	178	173	166	163	157	142	131	122	109	86	74	67	46	19	2
A3	217	215	210	201	198	190	170	154	153	135	105	89	82	55	30	6
A4	165	164	160	155	144	141	132	120	115	101	88	83	60	45	27	20
A5	248	245	236	235	224	220	205	179	172	140	115	107	80	77	40	11
A6	212	211	203	196	184	181	173	152	149	124	112	107	83	64	28	19
A7	235	233	229	227	215	205	191	185	145	137	115	95	61	53	31	5
A8	242	241	239	223	214	209	206	193	146	127	122	113	94	38	22	1
A9	245	238	235	230	206	202	197	190	155	146	139	133	85	70	59	10
A10	202	201	198	194	183	174	167	162	121	114	103	86	57	41	22	9

Table 2-7 Example solutions for Order-32 LLMICT with (a1…a16) less than 256.

Set	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}	a_{16}
B1	97	95	92	88	80	80	73	73	64	64	55	55	41	31	20	4
B2	125	122	115	109	106	106	94	94	83	83	67	67	62	50	29	10
B3	74	73	71	70	62	62	55	55	50	50	41	41	25	22	14	7
B4	111	110	106	97	90	90	79	79	78	78	65	65	54	33	15	2
B5	118	117	114	107	98	98	91	91	78	78	69	69	54	37	26	21
B6	105	103	95	93	87	87	81	81	67	67	59	59	49	45	21	5
B7	109	108	107	100	96	96	80	80	75	75	53	53	45	24	19	12
B8	115	110	109	98	94	94	89	89	77	77	67	67	61	38	35	10

Table 2-8 Example solutions for order-32 LLMICT with (a1…a16) less than 128.

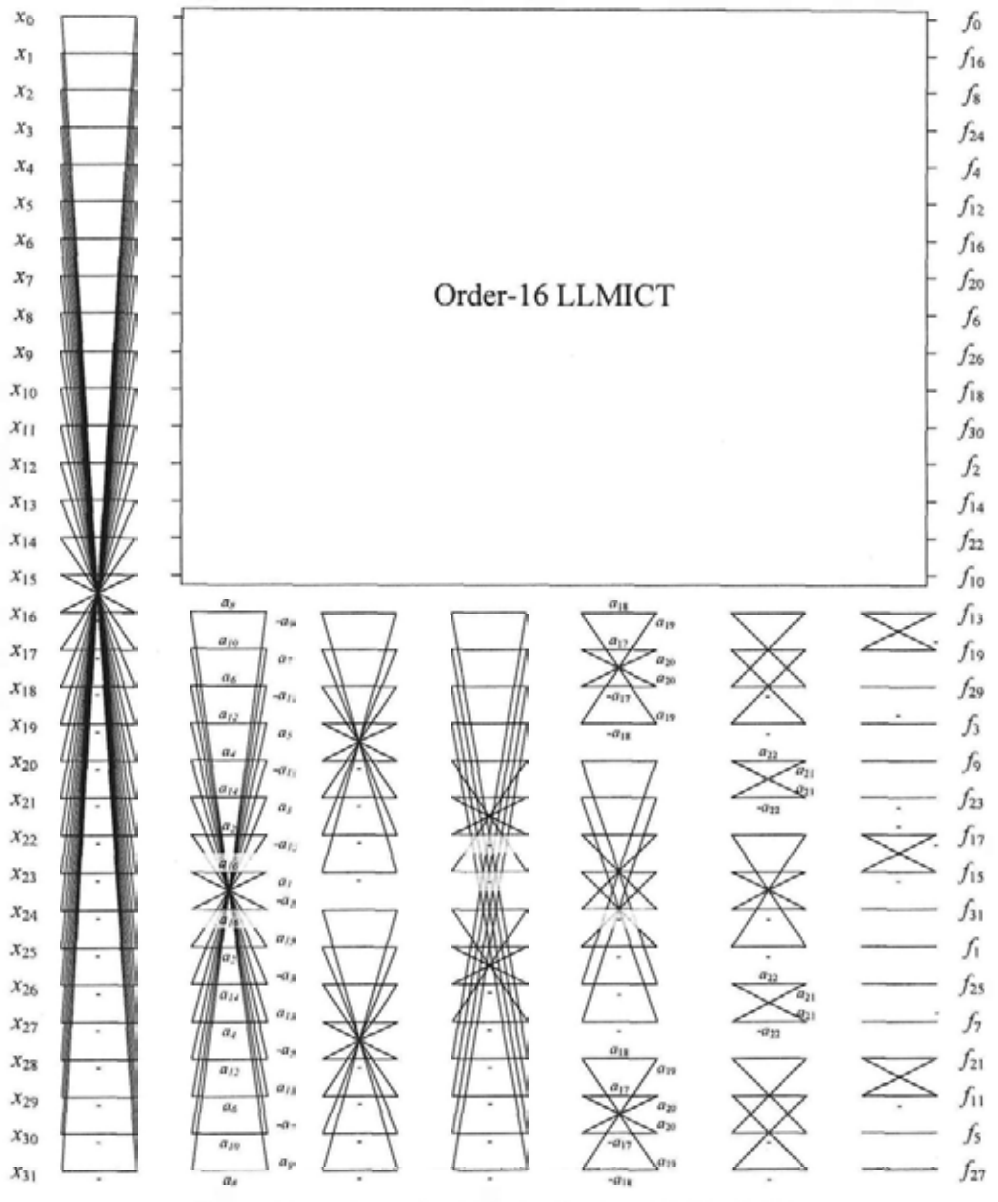


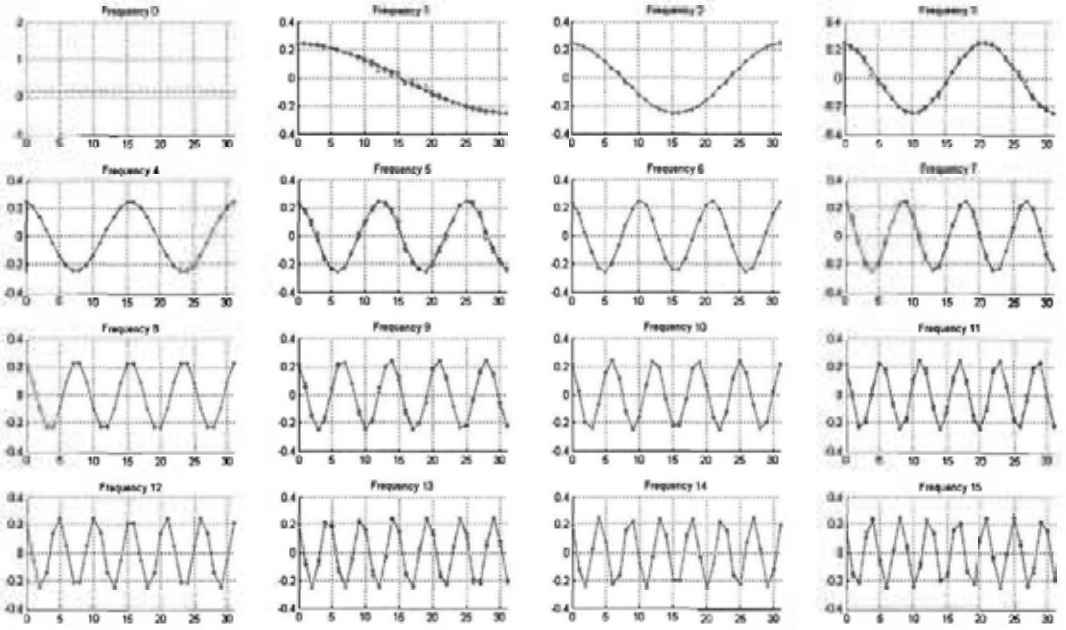
Figure 2-9 Fast Algorithm of order-32 LLMICT.

Take $(a_{17}, a_{18}, a_{19}, a_{20}) = (8, 7, 4, 1)$. One order-32 LLMICT A1 with $(a_1 \ldots a_{16}) = (253, 251, 243, 237, 233, 219, 197, 181, 177, 159, 127, 99, 89, 71, 33, 9)$ and another B1 with $(a_1 \ldots a_{16}) = (111, 110, 106, 97, 90, 90, 79, 79, 78, 78, 65, 65, 54, 33, 15, 2)$ are built. They are shown in Figure 2-11 and Figure 2-12 respectively. Their transform matrices are shown in the next two pages without normalizing their basis vector norms. It is shown that their matrix elements requires 12 bits and 11 bits

(including sign bit) to represent respectively. Order-32 LLMICT is still under investigation. Fine-tuning is required. It will not be included in the analysis in later sections and chapters. However, in Figure 2-10, it is shown that our proposed LLMICT have waveforms very close to the DCT. Here we show some analysis about these two transforms in Table 2-9. These figures are for reference only but they show that order-32 LLMICT are quite close to the order-32 DCT.

Transform	DCT Distortion (%)	Transform Coding Gain (dB)			
		$\rho = 0.6$	$\rho = 0.7$	$\rho = 0.8$	$\rho = 0.9$
Order-32 DCT	---	1.855	2.806	4.268	6.959
Order-32 LLMICT-A1	0.0136	1.847	2.792	4.246	6.921
Order-32 LLMICT-B1	0.0384	1.845	2.789	4.240	6.911

Table 2-9 Brief analysis of order-32 LLMICT



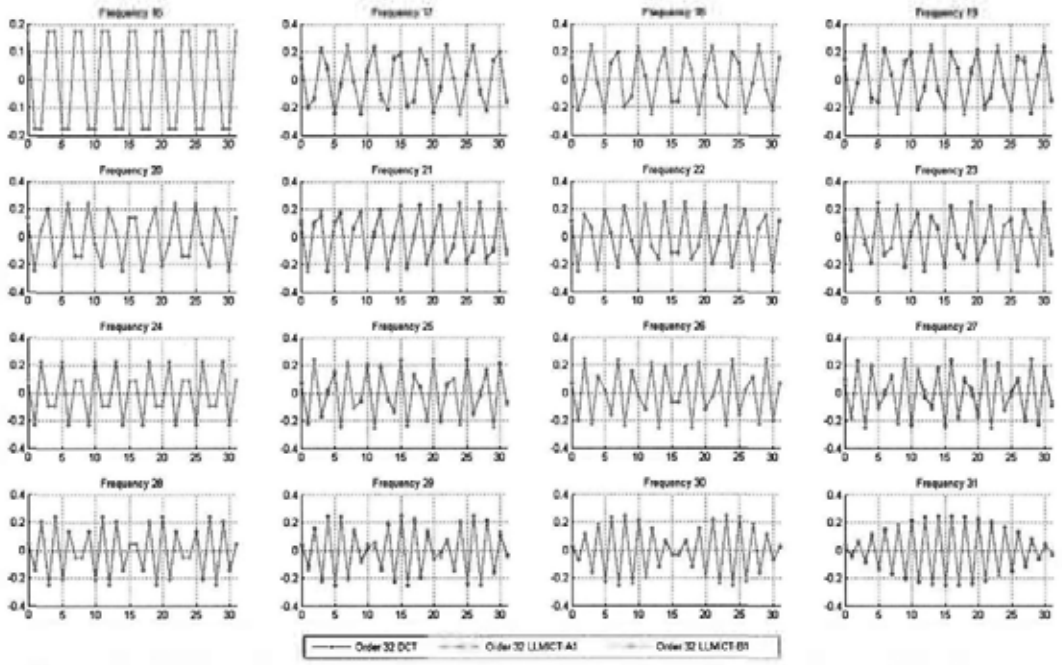


Figure 2-10 Waveforms of order-32 LLMICT and the DCT. The DCT is in blue, LLMICT-A1 is in red and LLMICT-B1 is in green.

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	...	1
179	178	173	166	163	157	142	131	122	109	86	74	67	46	19	2	-2	-19	...	-179
43	42	38	34	27	21	11	6	-6	-11	-21	-27	-34	-38	-42	-43	-43	-42	...	43
395	347	325	247	65	-65	-103	-215	-347	-395	-403	-403	-325	-247	-215	-103	103	215	...	-395
10	9	6	2	-2	-6	-9	-10	-10	-9	-6	-2	2	6	9	10	10	9	...	10
1963	1469	797	-239	-1303	-1873	-2041	-1807	-949	13	811	1571	2027	1879	1417	559	-559	-1417	...	-1963
319	203	33	-157	-291	-329	-261	-87	87	261	329	291	157	-33	-203	-319	-319	-203	...	319
1259	667	-197	-961	-1363	-1073	-337	551	1247	1321	841	-29	-967	-1349	-1189	-523	523	1189	...	-1259
5	2	-2	-5	-5	-2	2	5	5	2	-2	-5	-5	-2	2	5	5	2	...	5
1247	401	-841	-1363	-919	131	1189	1283	461	-667	-1357	-1007	29	1073	1303	551	-551	-1303	...	-1247
283	49	-261	-319	-87	203	327	171	-171	-327	-203	87	319	261	-49	-283	-283	-49	...	283
1261	-26	-1338	-1133	134	1342	1014	-429	-1378	-1027	531	1437	894	-541	-1443	-702	702	1443	...	-1261
9	-2	-10	-6	6	10	2	-9	-9	2	10	6	-6	-10	-2	9	9	-2	...	9
249	-66	-286	-39	234	234	-146	-281	-66	249	169	-169	-286	-39	281	146	-146	-281	...	-249
49	-31	-59	7	61	17	-53	-37	37	53	-17	-61	-7	59	31	-49	-49	31	...	49
181	-159	-219	99	237	-71	-251	9	253	33	-243	-89	233	127	-197	-177	177	197	...	-181
1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	...	1
177	-197	-127	233	89	-243	-33	253	-9	-251	71	237	-99	-219	159	181	-181	-159	...	-177
37	-53	-17	61	-7	-59	31	49	-49	-31	59	7	-61	17	53	-37	-37	53	...	37
146	-281	-39	286	-169	-169	249	66	-281	146	234	-234	-39	286	-66	-249	249	66	...	-146
6	-10	2	9	-9	-2	10	-6	-6	10	-2	-9	9	2	-10	6	6	-10	...	6
702	-1443	541	894	-1437	531	1027	-1378	429	1014	-1342	134	1133	-1338	26	1261	-1261	-26	...	-702
171	-327	203	87	-319	261	49	-283	283	-49	-261	319	-87	-203	327	-171	-171	327	...	171
551	-1303	1073	-29	-1007	1357	-667	-461	1283	-1189	131	919	-1363	841	401	-1247	1247	-401	...	-551
2	-5	5	-2	-2	5	-5	2	2	-5	5	-2	-2	5	-5	2	2	-5	...	2
523	-1189	1349	-967	29	841	-1321	1247	-551	-337	1073	-1363	961	-197	-667	1259	-1259	667	...	-523
87	-261	329	-291	157	33	-203	319	-319	203	-33	-157	291	-329	261	-87	-87	261	...	87
559	-1417	1879	-2027	1571	-811	13	949	-1807	2041	-1873	1303	-239	-797	1469	-1963	1963	-1469	...	-559
2	-6	9	-10	10	-9	6	-2	-2	6	-9	10	-10	9	-6	2	2	-6	...	2
103	-215	247	-325	403	-403	395	-347	215	-103	65	65	-247	325	-347	395	-395	347	...	-103
6	-11	21	-27	34	-38	42	-43	43	-42	38	-34	27	-21	11	-6	-6	11	...	6
2	-19	46	-67	74	-86	109	-122	131	-142	157	-163	166	-173	178	-179	179	-178	...	-2

Figure 2-11 the proposed order-32 LLMICT-A1

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	...	1
157	157	155	155	151	139	125	113	109	95	73	43	25	25	1	1	-1	-1	...	-157
43	42	38	34	27	21	11	6	-6	-11	-21	-27	-34	-38	-42	-43	-43	-42	...	43
865	710	655	370	10	-158	-230	-458	-769	-865	-881	-895	-815	-610	-545	-230	230	545	...	-865
10	9	6	2	-2	-6	-9	-10	-10	-9	-6	-2	2	6	9	10	10	9	...	10
862	703	430	95	-463	-815	-895	-785	-430	10	370	766	890	785	554	241	-241	-554	...	-862
319	203	33	-157	-291	-329	-261	-87	87	261	329	291	157	-33	-203	-319	-319	-203	...	319
551	239	-145	-505	-593	-464	-145	232	551	580	377	76	-320	-580	-548	-232	232	548	...	-551
5	2	-2	-5	-5	-2	2	5	5	2	-2	-5	-5	-2	2	5	5	2	...	5
548	232	-320	-580	-464	47	520	559	212	-295	-596	-377	145	505	551	239	-239	-551	...	-548
283	49	-261	-319	-87	203	327	171	-171	-327	-203	87	319	261	-49	-283	-283	-49	...	283
1103	-149	-1215	-795	303	1185	885	-355	-1215	-905	445	1229	985	-355	-1257	-621	621	1257	...	-1103
9	-2	-10	-6	6	10	2	-9	-9	2	10	6	-6	-10	-2	9	9	-2	...	9
1095	-165	-1265	-445	905	1039	-635	-1227	-311	1095	723	-885	-1185	-45	1255	635	-635	-1255	...	-1095
49	-31	-59	7	61	17	-53	-37	37	53	-17	-61	-7	59	31	-49	-49	31	...	49
158	-156	-180	130	194	-66	-220	4	222	30	-212	-108	180	130	-158	-156	156	158	...	-158
1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	...	1
156	-158	-130	180	108	-212	-30	222	-4	-220	66	194	-130	-180	156	158	-158	-156	...	-156
37	-53	-17	61	-7	-59	31	49	-49	-31	59	7	-61	17	53	-37	-37	53	...	37
635	-1255	-45	1185	-885	-723	1095	311	-1227	635	1039	-905	-445	1265	-165	-1095	1095	165	...	-635
6	-10	2	9	-9	-2	10	-6	-6	10	-2	-9	9	2	-10	6	6	-10	...	6
621	-1257	355	985	-1229	445	905	-1215	355	885	-1185	303	795	-1215	149	1103	-1103	-149	...	-621
171	-327	203	87	-319	261	49	-283	283	-49	-261	319	-87	-203	327	-171	-171	327	...	171
239	-551	505	-145	-377	596	-295	-212	559	-520	47	464	-580	320	232	-548	548	-232	...	-239
2	-5	5	-2	-2	5	-5	2	2	-5	5	-2	-2	5	-5	2	2	-5	...	2
232	-548	580	-320	-76	377	-580	551	-232	-145	464	-593	505	-145	-239	551	-551	239	...	-232
87	-261	329	-291	157	33	-203	319	-319	203	-33	-157	291	-329	261	-87	-87	261	...	87
241	-554	785	-890	766	-370	10	430	-785	895	-815	463	95	-430	703	-862	862	-703	...	-241
2	-6	9	-10	10	-9	6	-2	-2	6	-9	10	-10	9	-6	2	2	-6	...	2
230	-545	610	-815	895	-881	865	-769	458	-230	158	10	-370	655	-710	865	-865	710	...	-230
6	-11	21	-27	34	-38	42	-43	43	-42	38	-34	27	-21	11	-6	-6	11	...	6
1	-1	25	-25	43	-73	95	-109	113	-125	139	-151	155	-155	157	-157	157	-157	...	-1

Figure 2-12 the proposed order-32 LLMICT-B1

2.7 CSF Integer Cosine Transform

2.7.1 The CSF Fast DCT

Similar to the LLMICT, the proposed method can be applied to other fast DCT algorithms. For example, the fast DCT algorithm proposed by Chen et al [23]. In this thesis, it is named as CSF fast DCT algorithm. Similar to LLM fast DCT algorithm described in last section, it presents in a butterfly structure with multiplications with irrational constants. Its data flow is shown in Figure 2-13. Recently CSF fast DCT algorithm was proposed to be used in the Test Model under Consideration (TMuC, [28]) of JCT-VC to support larger transforms such as 16×16 and 32×32. However, the transform proposed in [28] is an integer approximation of the DCT such that it is not truly orthogonal. This motivates us to derive an orthogonal ICT with fast algorithm similar to CSF one. Here we call this transform as CSF Integer Cosine Transform (CSFICT).

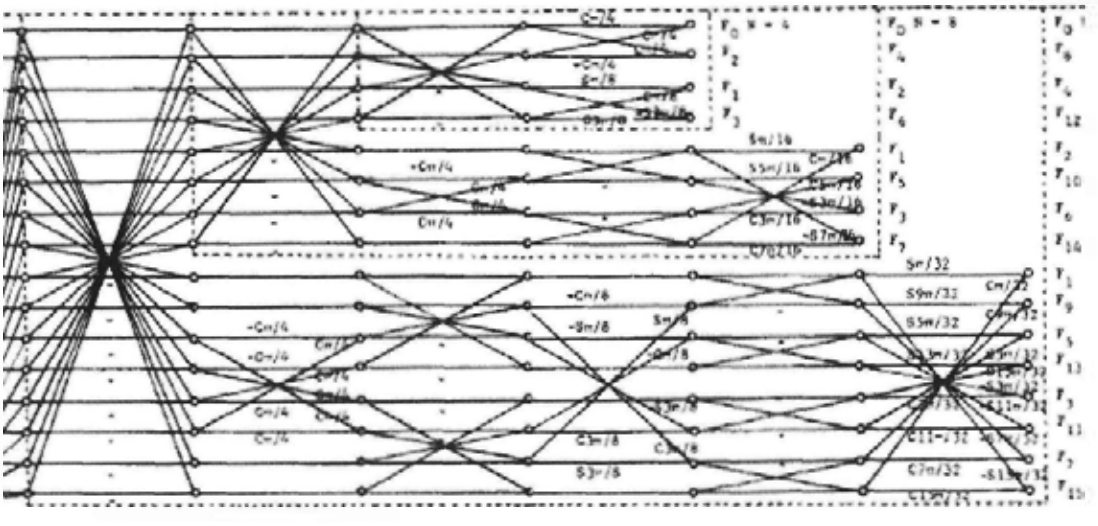


Figure 2-13 The CSF fast DCT algorithm.

2.7.2 CSF Integer Cosine Transform

Similar to LLMICT described in (2.50), the CSFICT is expressed as:

$$\mathbf{T}_{CSF}^{(2N)} = \begin{bmatrix} \frac{1}{\sqrt{2}}\mathbf{T}_{CSF}^{(N)} & \mathbf{0}_N \\ \mathbf{0}_N & \mathbf{P}_{CSF}^{(N)} \end{bmatrix} \begin{bmatrix} \mathbf{I}_N & \tilde{\mathbf{I}}_N \\ \mathbf{I}_N & -\tilde{\mathbf{I}}_N \end{bmatrix}. \tag{2.64}$$

For order-16 CSFICT, $\mathbf{P}_{CSF}^{(8)}$ can be found with order-16 CSF fast DCT algorithm. The odd part of the CSF fast DCT algorithm is generalized into Figure 2-14. In the figure, $a \ldots n$, α and β are real valued.

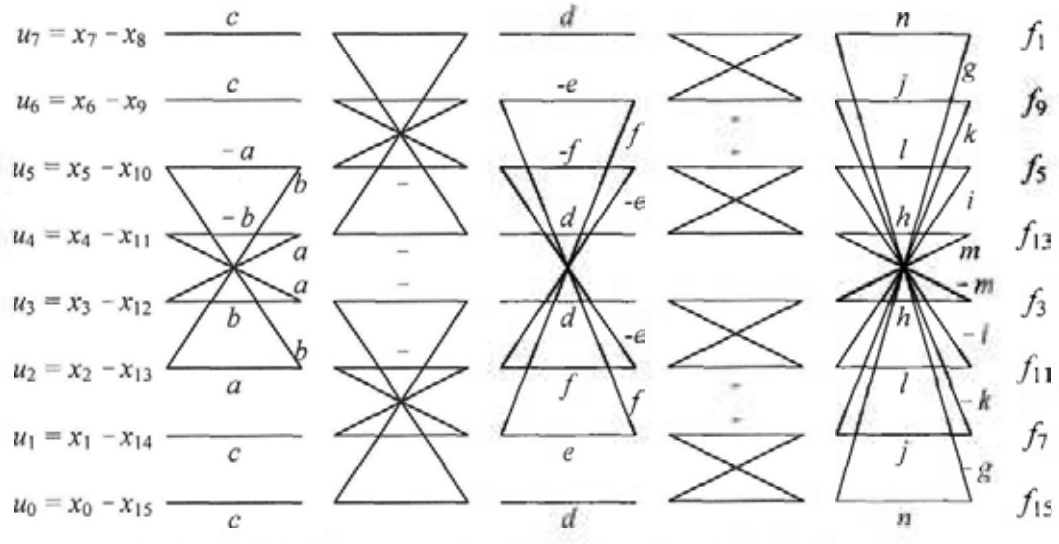


Figure 2-14 The generalized CSF fast algorithm (odd part).

It is expanded into matrix form:

$$\mathbf{P}_{CSF}^{(8)} =$$

$$\begin{bmatrix}
cdg & c(ge+nf) & a(ge+nf)-b(ne-gf) & d(na+gb) & d(ga-nb) & a(ne-gf)+b(ge+nf) & c(gf-ne) & cdn \\
cdh & c(me+hf) & -a(me+hf)+b(he-mf) & d(ma-hb) & -d(ha+mb) & -a(he-mf)-b(me+hf) & -c(he-mf) & -cdm \\
cdi & c(le-if) & -a(le-if)-b(ie+lf) & -d(la+ib) & -d(ia-lb) & a(ie+lf)-b(le-if) & c(ie+lf) & cdl \\
cdj & -c(je-kf) & -a(je-kf)-b(ke+jf) & -d(ka-jb) & d(ja+kb) & a(ke+kc)-b(je-kf) & -c(jf+ke) & -cdk \\
cdk & -c(jf+ke) & -a(ke+jf)+b(je-kf) & d(ja+kb) & d(ka-jb) & a(je-kf)-b(ke+jf) & c(je-kf) & cdj \\
cdl & -c(ie+lf) & a(ie+lf)-b(le-if) & d(ia-lb) & -d(la+ib) & a(le-if)+b(ie+lf) & c(le-if) & -cdi \\
cdm & -c(he-mf) & a(he-mf)+b(me+hf) & -d(ha+mb) & -d(ma-hb) & -a(hf+me)+b(he-mf) & -c(me+hf) & cdk \\
cdn & -c(gf-ne) & a(ne-gf)+b(ge+nf) & -d(ga-nb) & d(na+gb) & -a(ge+nf)+b(ne-gf) & c(ge+nf) & -cdg
\end{bmatrix} \tag{2.65}$$

The criteria for orthogonality are:

$$a^2 + b^2 = c^2 \text{ and} \tag{2.66}$$

$$e^2 + f^2 = d^2 \tag{2.67}$$

for arbitrary $g...n$. They are rather tough criteria if $a...f$ are limited to integers. Their values are found to be large so as to provide good performance. As a result, we proposed to modify the CSF fast DCT algorithm so as to loosen the criteria for orthogonality for CSFICT.

2.7.3 Modified CSF Fast DCT and MCSFICT

The odd part of CSF fast DCT algorithm is modified as shown Figure 2-15. This modified version is generalized to the one shown in Figure 2-16.

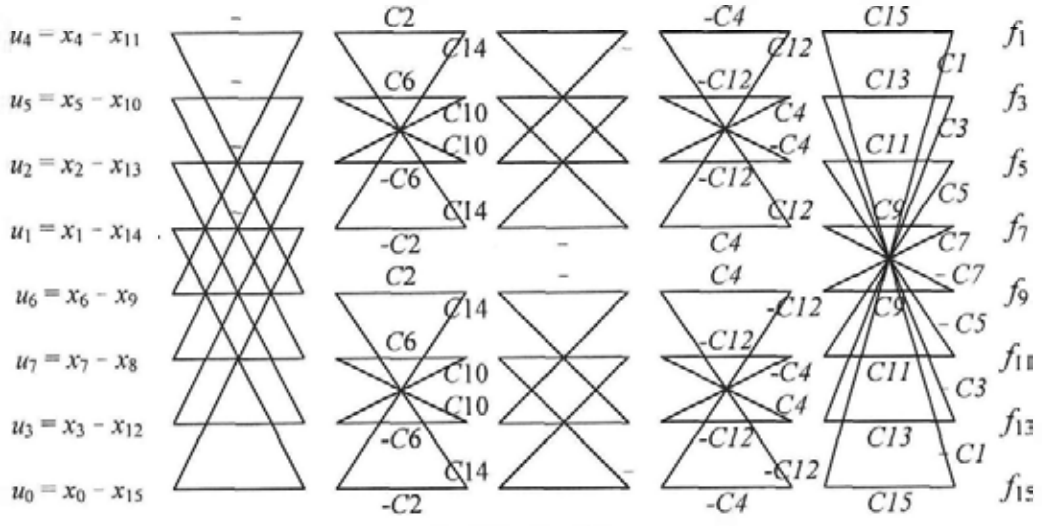


Figure 2-15 Modified CSF fast DCT algorithm (odd part).

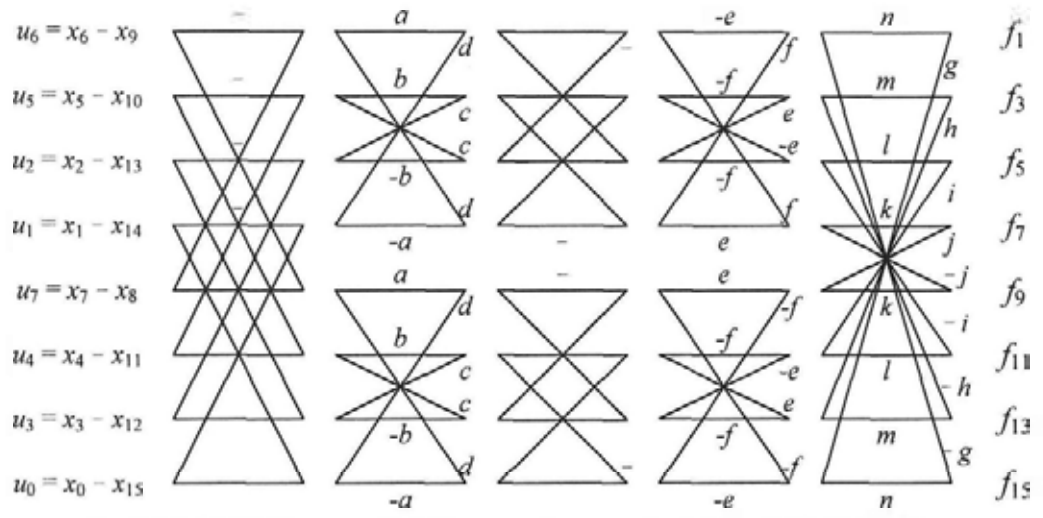


Figure 2-16 Generalized modified CSF fast algorithm (odd part).

The fast algorithm is expanded to matrix form similar to the one shown in (2.65) and the criterion for orthogonality is found:

$$a^2 + b^2 = c^2 + d^2 \tag{2.68}$$

for any real valued $e \ldots n$. (2.68) is a looser criterion with integer solution comparing with (2.66) and (2.67). Therefore, a simpler integer transform with good coding performance can be found easier. Here the integer transform found by the modified CSF fast algorithm is called MCSFICT. In order to make the MCSFICT have a good coding performance, the constants have to be:

$$a : b : c : d \approx \cos\left(\tfrac{\pi}{16}\right) : \cos\left(\tfrac{3\pi}{16}\right) : \cos\left(\tfrac{5\pi}{16}\right) : \cos\left(\tfrac{7\pi}{16}\right) \tag{2.69}$$

and

$$e : f \approx \cos\left(\tfrac{\pi}{8}\right) : \cos\left(\tfrac{3\pi}{8}\right) \tag{2.70}$$

Some suggested values are:

$$(a,b,c,d) \in \{(8,7,4,1),(11,9,7,3),(22,18,13,3),(19,16,11,4),...\} \text{ and} \tag{2.71}$$

$$(e,f) \in \{(2,1),(5,2),(7,3),(12,5),...\}. \tag{2.72}$$

After choosing the value of $a...f$, the remaining constants $g...n$ can be found by exhaustive search so that the highest coding performance is offered. The proposed method of deriving ICT has a high flexibility such that the values of $a...n$ can be chosen as a balance of complexity and performance. In this thesis, we suggest a MCSFICT with $a...n$ equal to (8, 7, 4, 1, 5, 2, 9, 8, 8, 7, 5, 4, 3, 1). The even part of this MCSFICT is taken to be $ICT_8(10, 9, 6, 2, 8, 4, 8)$. Its integer kernel matrix is:

$$\mathbf{E}_{MCSF} = \tag{2.73}$$

$$\begin{bmatrix}
256 & 256 & 256 & 256 & 256 & 256 & 256 & 256 & 256 & 256 & 256 & 256 & 256 & 256 & 256 & 256 \\
389 & 367 & 333 & 279 & 277 & 209 & 141 & 57 & -57 & -141 & -209 & -277 & -279 & -333 & -367 & -389 \\
360 & 324 & 216 & 72 & -72 & -216 & -324 & -360 & -360 & -324 & -216 & -72 & 72 & 216 & 324 & 360 \\
367 & 241 & 81 & -177 & -326 & -362 & -282 & -54 & 54 & 282 & 362 & 326 & 177 & -81 & 241 & -367 \\
360 & 144 & -144 & -360 & -360 & -144 & 144 & 360 & 360 & 144 & -144 & -360 & -360 & -144 & 144 & 360 \\
320 & 16 & -320 & -368 & -124 & 220 & 388 & 220 & -220 & -388 & -220 & 124 & 368 & 320 & -16 & -320 \\
324 & -72 & -360 & -216 & 216 & 360 & 72 & -324 & -324 & 72 & 360 & 216 & -216 & -360 & -72 & 324 \\
287 & -133 & -359 & -19 & 373 & 103 & -349 & -239 & 239 & 349 & -103 & -373 & 19 & 359 & 133 & -287 \\
256 & -256 & -256 & 256 & 256 & -256 & -256 & 256 & 256 & -256 & -256 & 256 & 256 & -256 & -256 & 256 \\
239 & -349 & -103 & 373 & 19 & -359 & 133 & 287 & -287 & -133 & 359 & -19 & -373 & 103 & 349 & -239 \\
216 & -360 & 72 & 324 & -324 & -72 & 360 & -216 & -216 & 360 & -72 & -324 & 324 & 72 & -360 & 216 \\
220 & -388 & 220 & 124 & -368 & 320 & 16 & -320 & 320 & -16 & -320 & 368 & -124 & -220 & 388 & -220 \\
144 & -360 & 360 & -144 & -144 & 360 & -360 & 144 & 144 & -360 & 360 & -144 & -144 & 360 & -360 & 144 \\
54 & -282 & 362 & -326 & 177 & 81 & -241 & 367 & -367 & 241 & -81 & -177 & 326 & -362 & 282 & -54 \\
72 & -216 & 324 & -360 & 360 & -324 & 216 & -72 & -72 & 216 & -324 & 360 & -360 & 324 & -216 & 72 \\
57 & -144 & 209 & -277 & 279 & -333 & 367 & -389 & 389 & -367 & 333 & -279 & 277 & -209 & 141 & -57
\end{bmatrix}$$

It is analyzed together with other order-16 DCT-like integer transforms described in this chapter in next sections.

2.8 Analysis

In the last sections, three types of order-16 DCT-like integer transforms have been proposed and demonstrated. Together with other order-16 transforms proposed in previous literatures, they are listed in Table 2-10. The performances of these transforms are going to be compared. The performance to be evaluated includes the complexity and the compressibility. The numbers of operations for 1-D fast transform are investigated and the computation times for 2-D fast transform are measured. Here the compressibility is in terms of transform efficiency and transform coding gain. These transforms are compared with the sub-optimal transform, the DCT. As actual compressibility in a coding system involves the performance of many other coding tools as well, the evaluation of the overall compressibility will be shown in later chapters.

Transform	Descriptions
$\mathbf{T}_{SICT}$	Simple ICT [14]
$\mathbf{T}_{MICT}$	Modified ICT [19]
$\mathbf{T}_{Wien}$	Integer Transform Proposed by Wien [16]
$\mathbf{T}_{Lee}$	Integer Transform Proposed by Lee [17]
$\mathbf{T}_{Joshi}$	Integer Transform Proposed by Joshi [18]
$\mathbf{T}_{SI\text{-}H264}$	Proposed Simple Integer Transform
$\mathbf{T}_{SI\text{-}AVS}$	Proposed Simple Integer Transform
$\mathbf{T}_{HI1}$	Proposed Hybrid Integer Transform from $\mathbf{E}_{DWW1}$
$\mathbf{T}_{HI2}$	Proposed Hybrid Integer Transform from $\mathbf{E}_{DWW2}$
$\mathbf{T}_{LLM\text{-}A1}$	Proposed LLMICT A1
$\mathbf{T}_{LLM\text{-}B1}$	Proposed LLMICT B1
$\mathbf{T}_{MCSF}$	Proposed MCSFICT

Table 2-10 List of Order-16 DCT-like transforms in this thesis.

2.8.1 Complexity Analysis

Fast algorithms are implemented (see Appendix B) and their numbers of operations are listed in Figure 2-17. It is shown that our proposed Simple Integer Transforms, $\mathbf{T}_{S1\text{-}H264}$ and $\mathbf{T}_{S1\text{-}AVS}$ require the least number of operations. Only 108 and 124 operations are required respectively. After them, our proposed LLMICT, $\mathbf{T}_{LLM\text{-}B1}$, $\mathbf{T}_{LLM\text{-}A1}$ and $\mathbf{T}_{MCSF}$, follow. The numbers of operations are 140, 160 and 172 respectively. The remaining are $\mathbf{T}_{MICT}$, $\mathbf{T}_{Wien}$, $\mathbf{T}_{Joshi}$, $\mathbf{T}_{HI2}$, $\mathbf{T}_{HI1}$, and $\mathbf{T}_{Lee}$ in ascending order.

The number of computation is only a theoretical figure. In order to show a more realistic scene, these fast algorithms are tested with 1,000,000 sets of 16×16 random pixels. The testing platform is a PC with an Intel Core 2 Duo @ 2.53GHz and 2 GB memory working with Windows Vista. Their computation times are recorded and listed in Table 2-12. Comparing with direct matrix multiplication, these fast algorithms can speed up the process by 30 to 45 times. As shown in Figure 2-18, the two proposed Simple Integer Transforms $\mathbf{T}_{S1\text{-}AVS}$ and $\mathbf{T}_{S1\text{-}H264}$ are the two fastest transforms. They require less than 1.4 seconds completing 1,000,000 2-D transform operations. They are followed by our proposed LLMICT, $\mathbf{T}_{LLM\text{-}B1}$ and $\mathbf{T}_{LLM\text{-}A1}$. Their computation times slightly increase to 1.46 seconds. Our proposed $\mathbf{T}_{MCSF}$ can also complete the test in 1.52 seconds. The remaining are $\mathbf{T}_{Joshi}$, $\mathbf{T}_{Lee}$, $\mathbf{T}_{MICT}$, $\mathbf{T}_{Wien}$, $\mathbf{T}_{HI2}$ and $\mathbf{T}_{HI1}$. They take more than 1.6 seconds.

Transform	# of Add	# of Shift	Total # of Operation
$\mathbf{T}_{MICT}$	144	30	174
$\mathbf{T}_{Wien}$	144	38	182
$\mathbf{T}_{Lee}$	148	58	206
$\mathbf{T}_{Joshi}$	120	64	184
$\mathbf{T}_{S1\text{-}H264}$	80	28	108
$\mathbf{T}_{S1\text{-}AVS}$	92	32	124
$\mathbf{T}_{HI1}$	160	30	190
$\mathbf{T}_{HI2}$	158	30	188
$\mathbf{T}_{LLM\text{-}A1}$	110	50	160
$\mathbf{T}_{LLM\text{-}B1}$	92	48	140
$\mathbf{T}_{MCSF}$	114	58	172

Table 2-11 Number of operations for different order-16 transform (1-D)

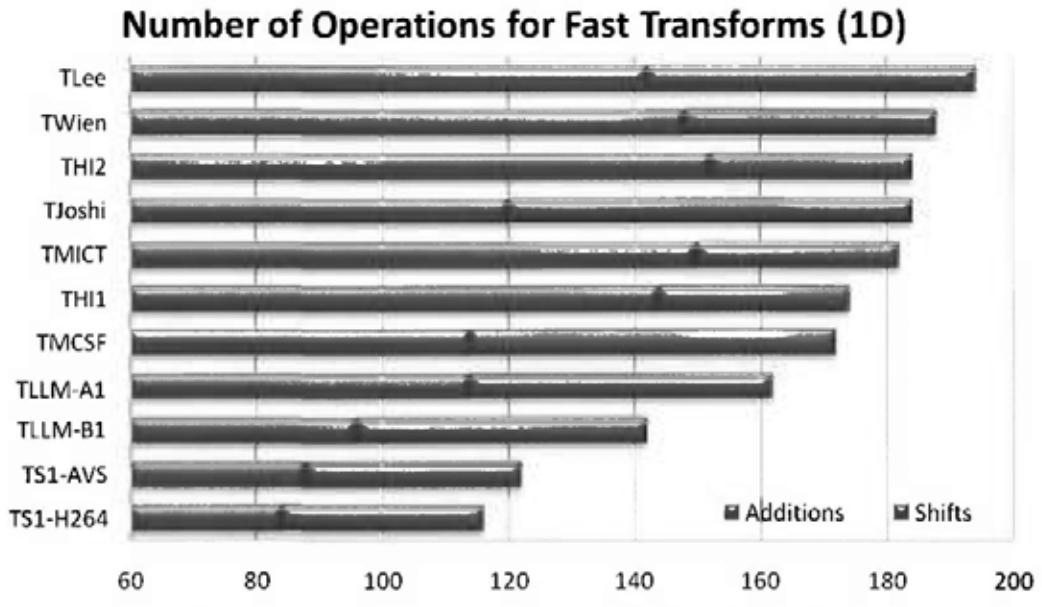


Figure 2-17 The numbers of operations of different fast transforms.

Transform	Direct Multiplication, t_{DM} (s)	Fast Algorithm, t_{FA} (s)	Speed-up $= \frac{t_{DM} - t_{FA}}{t_{FA}}$
$\mathbf{T}_{MICT}$	60.126	1.732	33.71
$\mathbf{T}_{Wien}$		1.829	31.87
$\mathbf{T}_{Lee}$		1.727	33.82
$\mathbf{T}_{Joshi}$		1.646	35.53
$\mathbf{T}_{S1\text{-}H264}$		1.355	43.37
$\mathbf{T}_{S1\text{-}AVS}$		1.304	45.11
$\mathbf{T}_{HI1}$		2.053	28.29
$\mathbf{T}_{HI2}$		1.999	29.08
$\mathbf{T}_{LLM\text{-}A1}$		1.460	40.18
$\mathbf{T}_{LLM\text{-}B1}$		1.460	40.18
$\mathbf{T}_{MCSF}$		1.520	38.56

Table 2-12 Computation time for different order-16 transform

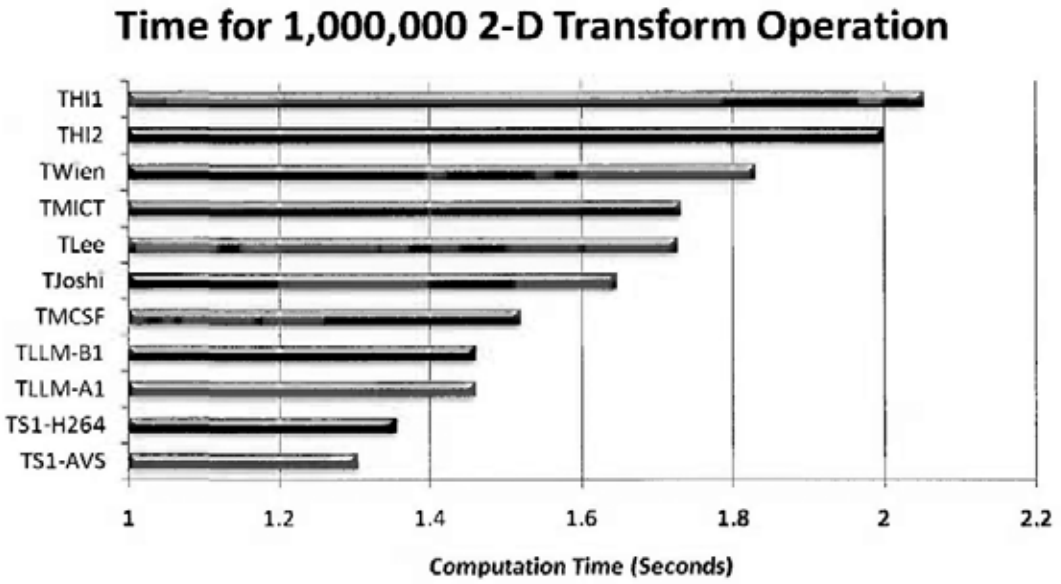


Figure 2-18 The computation time of different fast transforms.

2.8.2 DCT Distortion and Transform Efficiency

It is expected that a transform with basis vectors closer to the DCT basis vectors, the better coding performance will be. Here the DCT distortion discussed in [16] will be measured. If a transform has a lower DCT distortion, it is closer to the DCT. The DCT distortion is defined as:

$$d_2 = 1 - \frac{1}{16}\left\| diag\left(\mathbf{T}_{DCT}\,\mathbf{T}_{UT}^{T}\right)\right\|_2^2. \tag{2.74}$$

$\mathbf{T}_{DCT}$ is the order-16 DCT transform matrix and $\mathbf{T}_{UT}$ is the order-16 transform under test. If $\mathbf{T}_{UT}$ is the same as $\mathbf{T}_{DCT}$, d_2 is zero. If $\mathbf{T}_{UT}$ is closer to $\mathbf{T}_{DCT}$, the smaller the d_2 is. The DCT distortions of different transforms are shown in the 2nd column of Table 2-13. $\mathbf{T}_{Joshi}$ has the lower DCT distortion. $\mathbf{T}_{LLM\text{-}A1}$ is not far from it. It is followed by $\mathbf{T}_{MCSF}$, $\mathbf{T}_{SICT}$, $\mathbf{T}_{LLM\text{-}B1}$ and $\mathbf{T}_{Lee}$. These five transforms have DCT distortions lower than 1%. $\mathbf{T}_{HI1}$, $\mathbf{T}_{MICT}$, $\mathbf{T}_{HI2}$ and $\mathbf{T}_{Wien}$ have higher distortions of around 16%. The distortions of $\mathbf{T}_{SI\text{-}H264}$ and $\mathbf{T}_{SI\text{-}AVS}$ are the largest (>27%).

The efficiency of a transform is generally measured by its ability to decorrelate the pixel data. The transform efficiency is defined as:

$$\eta = \frac{\sum_{i=0}^{N-1}|b(i,i)|}{\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}|b(i,j)|} \quad \text{where} \quad \mathbf{T}_{UT}\mathbf{R}_{xx}\mathbf{T}_{UT}^{T} = \mathbf{B} = \{b(i,j)\}. \tag{2.75}$$

$\mathbf{R}_{xx}$ is the correlation matrix of the pixel data. Assume that it is a first order Markov process. The $(i, j)^{th}$ element of $\mathbf{R}_{xx}$ is $\rho^{|i-j|}$. It is reported that the correlation of the predicted residue (ρ) is ranged from 0.5 to 0.9 [19]. The

transform efficiency in this range is shown in Table 2-13. They are plotted against the correlation in Figure 2-19. The optimal transform, KLT, can completely decorrelate the pixel data such that 100% transform efficiency is achieved. It is shown that the DCT has the highest transform efficiency. After that are $\mathbf{T}_{LLM\text{-}A1}$ and $\mathbf{T}_{Joshi}$. They have almost the same efficiency. The remaining are $\mathbf{T}_{MCSF}$, $\mathbf{T}_{SICT}$, $\mathbf{T}_{LLM\text{-}B1}$, $\mathbf{T}_{Lee}$, $\mathbf{T}_{MICT}$, $\mathbf{T}_{HI2}$, $\mathbf{T}_{HI1}$, $\mathbf{T}_{S1\text{-}AVS}$, $\mathbf{T}_{S1\text{-}H264}$ and $\mathbf{T}_{Wien}$ in descending order.

Transform	DCT Distortion (%)	Transform Efficiency (%)				
		$\rho = 0.5$	$\rho = 0.6$	$\rho = 0.7$	$\rho = 0.8$	$\rho = 0.9$
$\mathbf{T}_{DCT}$	0.00	79.8	78.2	77.4	78.3	82.8
$\mathbf{T}_{SICT}$	0.23	78.0	76.0	74.9	75.1	79.4
$\mathbf{T}_{MICT}$	16.68	72.5	70.9	70.8	72.7	78.8
$\mathbf{T}_{Wien}$	16.95	60.5	56.8	54.1	53.8	59.9
$\mathbf{T}_{Lee}$	0.80	75.7	73.4	72.0	72.6	77.2
$\mathbf{T}_{Joshi}$	0.05	79.1	77.4	76.5	77.2	81.6
$\mathbf{T}_{S1\text{-}H264}$	27.52	64.3	62.6	62.6	65.1	72.8
$\mathbf{T}_{S1\text{-}AVS}$	28.26	63.8	61.7	61.1	62.9	70.1
$\mathbf{T}_{HI1}$	16.49	71.1	69.3	69.1	71.2	78.2
$\mathbf{T}_{HI2}$	16.79	71.5	69.7	69.4	71.3	77.7
$\mathbf{T}_{LLM\text{-}A1}$	0.07	79.4	77.6	76.6	77.2	81.3
$\mathbf{T}_{LLM\text{-}B1}$	0.50	75.6	73.5	72.6	73.4	78.5
$\mathbf{T}_{MCSF}$	0.20	78.9	76.9	75.7	76.0	80.2

Table 2-13 DCT Distortion and Transform Efficiency.

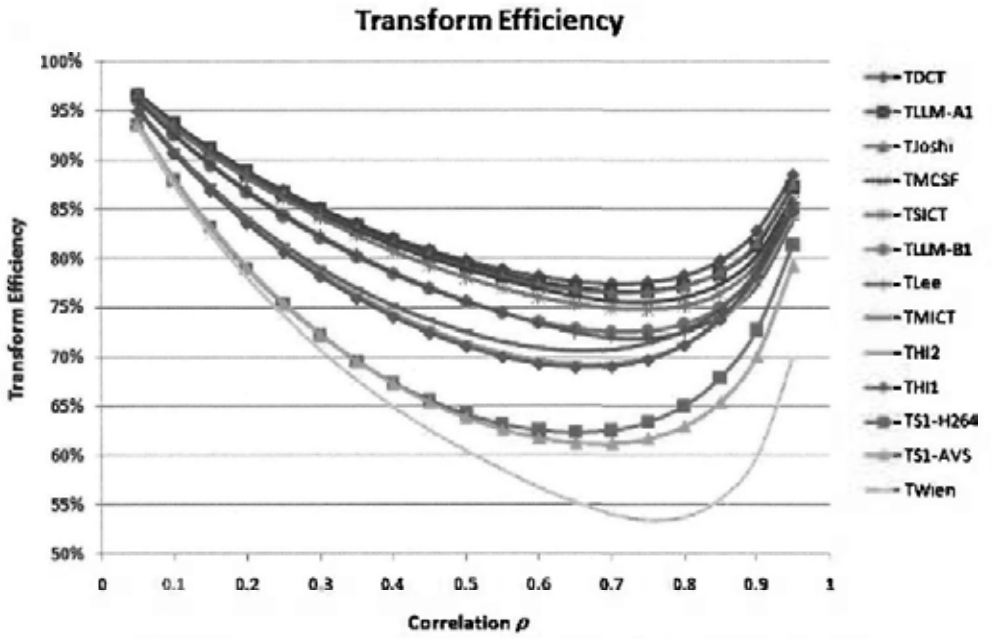


Figure 2-19 The transform efficiency of different transforms.

2.8.3 Transform Coding Gain

Another important measure for the evaluation of the transform performance is the transform coding gain G_{TC}. Under the assumptions of optimum quantization and bit allocation, G_{TC} of an order-N transform is:

$$G_{TC} = \frac{1}{N}\sum_{k=0}^{N-1}\sigma_k^2 \Big/ \left(\prod_{k=0}^{N-1}\sigma_k^2\right)^{\frac{1}{N}} \tag{2.76}$$

where $\sigma_k^2 = b(k,k)$ is the variance of the k-th transform coefficient (recall $b(k, \bar{k})$ in (2.75)). The transform coding gain at different ρ is shown in Table 2-14. As the G_{TC} are quite close to each other, their differences between the DCT are plotted in Figure 2-20. Again, $\mathbf{T}_{LLM\text{-}A1}$ and $\mathbf{T}_{Joshi}$ are very close to the DCT. They are lower than the DCT not more than 0.02dB which is negligible. After them, $\mathbf{T}_{MCSF}$, $\mathbf{T}_{SICT}$ and $\mathbf{T}_{LLM\text{-}B1}$ follow. Both lag behind the DCT not more than 0.1dB. The remaining are $\mathbf{T}_{Lee}$, $\mathbf{T}_{MICT}$, $\mathbf{T}_{HI1}$, $\mathbf{T}_{HI2}$, $\mathbf{T}_{S1\text{-}AVS}$, $\mathbf{T}_{S1\text{-}H264}$ and $\mathbf{T}_{Wien}$ in descending order.

Transform	Transform Coding Gain, G_{TC} (dB)				
	$\rho = 0.5$	$\rho = 0.6$	$\rho = 0.7$	$\rho = 0.8$	$\rho = 0.9$
$\mathbf{T}_{DCT}$	1.141	1.779	2.698	4.115	6.726
$\mathbf{T}_{SICT}$	1.135	1.769	2.682	4.089	6.685
$\mathbf{T}_{MICT}$	1.068	1.679	2.569	3.957	6.540
$\mathbf{T}_{Wien}$	0.871	1.367	2.100	3.281	5.602
$\mathbf{T}_{Lee}$	1.112	1.732	2.622	3.995	6.539
$\mathbf{T}_{Joshi}$	1.139	1.775	2.692	4.106	6.711
$\mathbf{T}_{S1\text{-}H264}$	0.932	1.484	2.307	3.619	6.118
$\mathbf{T}_{S1\text{-}AVS}$	0.929	1.478	2.296	3.600	6.088
$\mathbf{T}_{HI1}$	1.064	1.672	2.559	3.942	6.518
$\mathbf{T}_{HI2}$	1.066	1.674	2.561	3.943	6.515
$\mathbf{T}_{LLM\text{-}A1}$	1.141	1.778	2.695	4.108	6.712
$\mathbf{T}_{LLM\text{-}B1}$	1.121	1.749	2.654	4.052	6.631
$\mathbf{T}_{MCSF}$	1.136	1.771	2.684	4.094	6.692

Table 2-14 Transform coding gain of different transform.

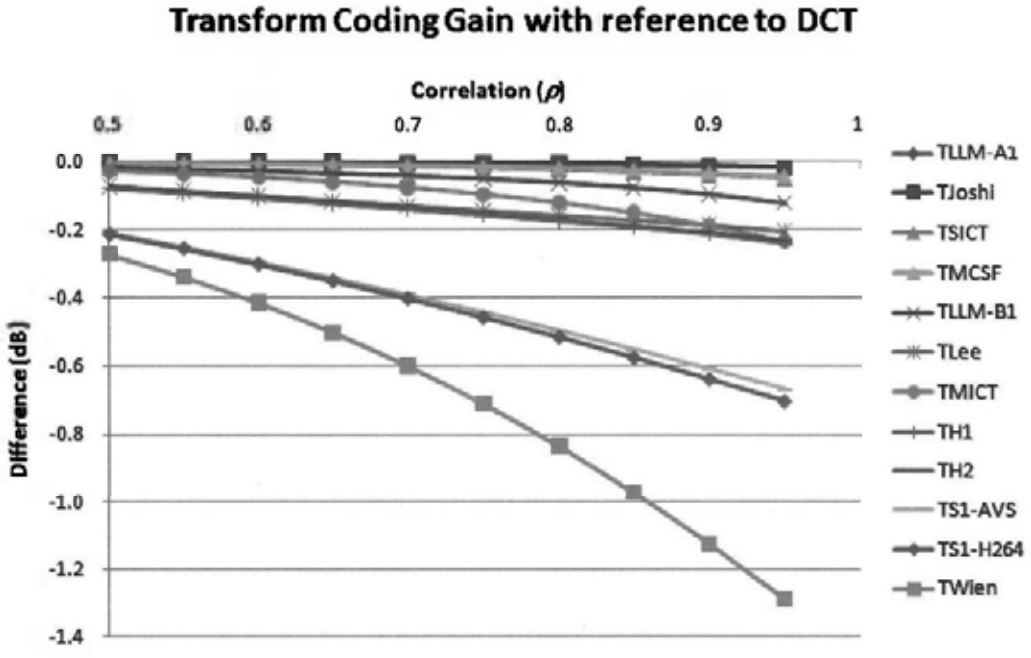


Figure 2-20 The transform coding gains of different transforms (reference to DCT).

2.8.4 Computationally Optimal Transform

In any encoding situation, including image, video and audio, rate-distortion optimization is a common technique to minimize the distortion (or to maximize the quality) under a given data rate. This is very well-known that the output quality will be higher when a higher data rate is offered. Just like the relationship between rate and distortion, the computation and the coding performance also have similar relationship. It is very intuitive that the transform with more computation (i.e. the more precise approximation of the DCT or KLT), the higher coding gain can be obtained. However, which one is the computationally optimal one? The computationally optimal transform can be found by the method similar to rate-distortion (RD) curve. The coding gain is plotted against the number of operations as shown in Figure 2-21. When the transform is by-passed, the gain is 1 (or 0 dB). When the number of operations in the transform increases, the transform can be implemented more precisely. As a result, it should be approaching to the optimal transform, KLT. The coding gain will approach to that of the KLT (the green dotted-dashed line). An operational complexity-coding-gain (CCG) curve, which is the upper envelope of all complexity-coding gain points, can be found. It is the red dotted line as shown. The transform has a complexity-coding gain points closer to the CCG curve implying that it is more computationally efficient. It can be observed that $\mathbf{T}_{LLM\text{-}A1}$ and $\mathbf{T}_{Joshi}$ are the two most computationally optimal transform among the tested transforms theoretically. They are very close to the operational CCG curve with high coding gain. With the CCG curve, we can select different transform according to the desired application. For example, we can choose $\mathbf{T}_{SI\text{-}H264}$ for application requires low complexity.

The proposed order-16 integer transforms and existing integer transforms are described and tested theoretically in this chapter. In the next two chapters, these transforms are integrated into the reference software of two popular video coding standards, H.264/AVC and AVS in China. An overview of each standard will be given. The integration method will be described in detail. Experiment result of the proposed platforms will be shown. The performance of these integer transforms will be compared.

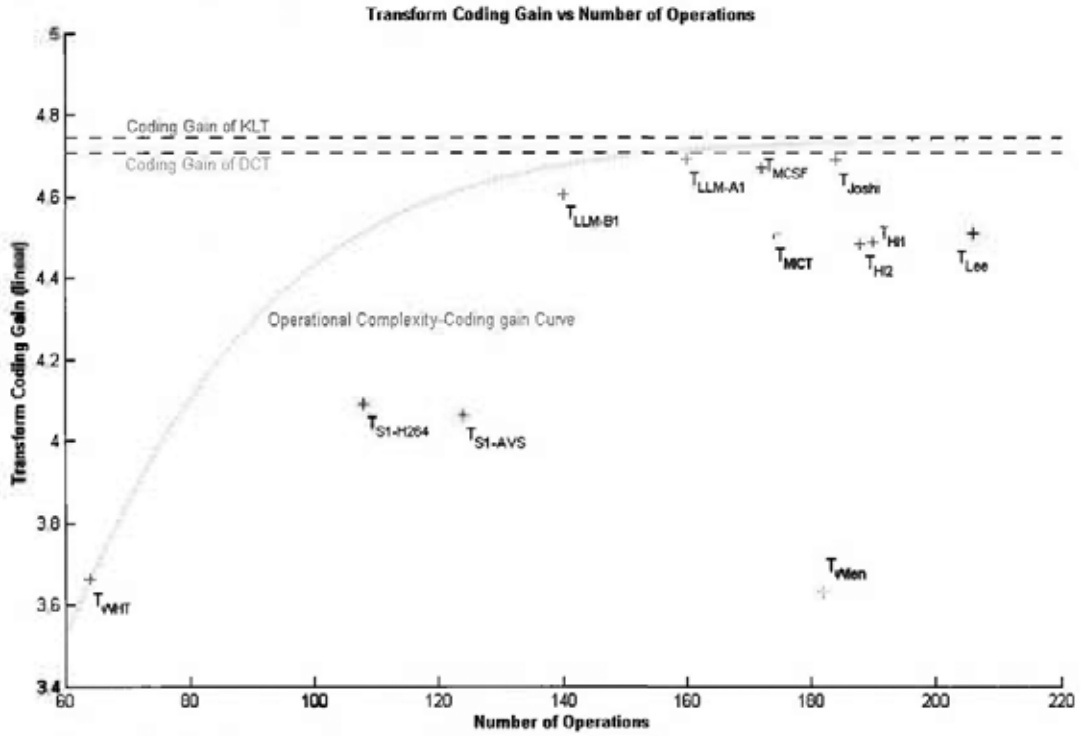


Figure 2-21 The transform coding gain ($\rho = 0.9$) vs. the number of operation.

2.9 Conclusions

In this chapter, 3 classes of order-16 transforms are proposed. They include (i) the Simple Integer Transform extended from existing order-8 ICT, (ii) the Hybrid Integer Transform formed by order-8 ICT and order-8 Dyadic Weighted Walsh Transform and (iii) the order-16 ICT derived from Relaxed General Cosine Transform (RGCT). The third class of transforms can be further divided into two types: LLMICT and CSFICT. These two types are very similar to the DCT. The order-32 LLMICT are also shown. CSFICT is modified to loosen its criterion for orthogonality. This increases the flexibility of designing high performance orthogonal transform. Examples of these order-16 transforms are shown and investigated. The simple integer transform is the simplest transform. The two proposed simple integer transforms only take 110~120 operations to complete a single 1-D transform. Experiment shows that they take around 1.3 seconds to complete 1,000,000 times 2-D transform. The proposed LLMICT has the highest coding gain. The waveforms of the two proposed LLMICT are very close to the DCT. Their DCT distortions are 0.07% and 0.50% respectively only. At the same time, they do not require complex computation such that their computation times are only slightly longer than that of simple integer transform. The ultimate transform should be computationally optimal such that it is high compressibility and low computation requirement at the same time. In order to find the computationally optimal transform, the operational computational-gain curve is proposed. It is found that $\mathbf{T}_{LLM\text{-}A1}$ and $\mathbf{T}_{Joshi}$ are the two most computationally optimal transforms in theory.

2.10 References

[1] W.-K. Cham, "*Development of integer cosine transforms by the principle of dyadic symmetry,*" IEE Proc. I, vol. 136, issue 4, pp. 276-282, 1989.

[2] N. Ahmed, T. Natarajan and K. R. Rao, "Discrete Cosine Transform," IEEE Trans. on Computers, vol. C-23, issue 1, pp. 90-93, 1974.

[3] N. S. Jayant and P. Noll, Digital Coding of Waveforms: Principles and Applications to Speech and Video. Englewood Cliffs, NJ: Prentice-Hall, 1984.

[4] "*Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification,*" document JVT-G050, ITU-T Rec. H.264 and ISO/IEC 14496-10 AVC, Mar. 2003, [Online]. Available: http://ftp3.itu.ch/avarch/jvt-site.

[5] Advanced Video Coding for Generic Audiovisual Services, ITU-T Rec. H.264 and ISO/IEC 14496-10 AVC, Mar. 2005.

[6] Information Technology-Advanced Coding of Audio and Video-Part 2: Video, GB/T 20090.2-2006 AVS, May 2006.

[7] Jay Loomis and Mike Wasson, "*VC-1 Technical Overview,*" Microsoft Corporation, Oct. 2007. [Online]

[8] M. Wien, "*Variable block-size transforms for H.264/AVC,*" IEEE Trans. on CASVT, vol. 13, issue 7, pp. 604-613, 2003.

[9] Steve Gordon, Detlev Marpe and Thomas Wiegand, "*Simplified Use of 8x8 Transforms – Updated Proposal and Results,*" document JVT-K028, March 2004 [online]. Available: http://ftp3.itu.ch/av-arch/jvt-site/2004_03_Munich/

[10] D. Marpe, T. Wiegand and S. Gordon, "*H.264/MPEG4-AVC fidelity range extensions: tools, profiles, performance and application areas,*" IEEE ICIP 2005, vol. 1, pp 1 - 593-596, 2005.

[11] T. Sikora and B. Makai, "*Shape-adaptive DCT for generic coding of video,*" IEEE Trans on CASVT, vol. 5, issue 1, pp 59-62, 1995.

[12] Yan Ye and M. Karczewicz, "*Improved H.264 intra coding based on bi-directional intra prediction, directional transform, and adaptive coefficient scanning,*" 15th IEEE ICIP, pp 2116-2119, 2008.

[13] Y. Zhou, L. Zhang and S. W. Ma, "*Rate-Distortion Optimized Transform,*" Document AVS-M2676, March 2010. [Chinese]

[14] W. K. Cham and Y. T. Chan, "*An order-16 integer cosine transform,*" IEEE Trans. Signal Process., vol. 39, no. 5, pp. 1205–1208, May 1991.

[15] S. N. Koh, S. J. Huang and H. K. Tang, "*Development of order-16 integer transforms,*" Signal Processing, Vol. 24, Issue 3, pp. 283-289, Sept 1991.

[16] Mathias Wien and Shijun Sun, "*ICT Comparison for Adaptive Block Transforms,*" document VCEG-L12, Jan., 2001. [Online] Available: http://wftp3.itu.int/av-arch/video-site/0101_Eib

[17] Bumshik Lee and Munchurl Kim, "*A 16×16 transform kernel with quantization for (ultra) high definition video coding,*" document VCEG-AK13, April 2009. [Online] Available: http://wftp3.itu.int/av-arch/video-site/0904_Yok/

[18] R. Joshi, Y. Reznik, and M. Karczewicz, "*Simplified Transforms for Extended Block Sizes,*" document VCEG-AL30, July 2009. [Online]

Avaliable: http://wftp3.itu.int/av-arch/video-site/0906_LG/

[19] Jie Dong; King Ngi Ngan; Chi-Keung Fong; Wai-Kuen Cham, "*2-D Order-16 Integer Transforms for HD Video Coding,*" IEEE Trans. on CASVT, vol. 19, Issue: 10, pp. 1462 – 1474, Oct. 2009.

[20] Chi-Keung Fong and Wai-Kuen Cham, "*Simple Order-16 Integer Transform for Video Coding,*" to be published in ICIP 2010.

[21] X. Mao, Y. Wang, Y. He, W.K. Cham, C.K. Fong, J. Dong, K.N. Ngan, H. M. Wong, L. Wang, Y. Huo, T. Pun, C. Cheng, "*AVS Adaptive Block-size Transform,*" AVS Video Proposal AVS-M2372, Xiamen, June 2008.

[22] W. K. Cham and R. J. Clarke, "*Dyadic Symmetry and Walsh matrices,*" IEE Proc. Commun., Radar, and Signal Process., vol. 134, pp. 141 – 144, 1987.

[23] W. H. Chen, C. H. Smith, and S. C. Fralick, "*A Fast Computational Algorithm for the Discrete Cosine Transform,*" IEEE Trans. on Communication, vol. COM-25, no. 9, pp.1004-1009, Sep. 1977.

[24] B. G. Lee, "*A New Algorithm to Compute the Discrete Cosine Transform,*" IEEE Trans. on Acoust., Speech, Signal Processing, vol. ASSP-32, no. 6, pp.1243-1245, Dec. 1984.

[25] H. S. Hou, "*A Fast Recursive Algorithm For Computing the Discrete Cosine Transform,*" IEEE Trans. on Acoust., Speech, Signal Processing, vol. ASSP-35, No. 10, pp.1455-1461, Oct. 1987.

[26] Loeffler C., Ligtenberg A., Moschytz C.S., "*Practical Fast 1D DCT Algorithm with Eleven Multiplications,*" Proc. ICASSP, pp. 988-991, 1989.

[27] C. W. Kok, "*Fast Algorithm for Computing Discrete Cosine Transform,*" IEEE Trans. on Signal Process., vol. 45, no. 3, pp.757-760, Mar. 1997.

[28] "Suggestion for a Test Model," document JCTVC-A033, April 2010.

Chapter 3 ABT in H.264/AVC

3.1 Overview of H.264/AVC

H.264/AVC (also called H.264/MPEG-4 Part 10) [1][2] is a video coding standard jointly developed by the ITU-T Video Coding Experts Group (VCEG) and the ISO/IEC Moving Picture Experts Group (MPEG). These two parties formed the Joint Video Team (JVT) and developed this standard. This standard targets in the applications including video storage, streaming video on web, digital video broadcast and real-time video-conferencing. It replaces the old video coding standard MPEG-2. Many factors, such as new algorithms, the dramatic rise of the processor speed and the fall of the memory cost during the last decade, break the development constraints in the past. Computationally complex but functionally efficient algorithms are now feasible. H.264/AVC adopted many advanced algorithms into its block-based hybrid video coding structure to improve its coding performance. As a result, it is reported that H.264/AVC reduces the bit rate by half at the same objective quality comparing

with MPEG-2.

The JVT extended the H.264/AVC standard and named these extensions as Fidelity Range Extensions (FREXt) [3][4]. They improve the quality of video coding. It supports higher sample bit depth precision, higher color sub-sampling rates (such as 4:2:2 and 4:4:4), addition of 8×8 ICT and many other new features. The drafting work on these extensions was completed in 2004. Besides FREXt, other main features such as Scalable Video Coding (SVC, completed in 2007) [5]-[8] and Multiview Video Coding (MVC, completed in 2009) [9]-[11] are also added to the standard. However, these two features are not in the scope of this thesis. Their detail will not be discussed here.

In Figure 3-1 and Figure 3-2, the diagrams of the encoder and the decoder of H.264/AVC are shown respectively. It is a typical hybrid video coding structure. The spatial, temporal and statistical redundancies in the video sequence are removed in different stages in the encoder. The intra prediction and integer transform remove the spatial redundancies. The motion estimation with multiple reference frames removes the temporal redundancies. The entropy coders (Context-Adaptive Variable Length Coder, CAVLC and Context-based Adaptive Binary Arithmetic Coder, CABAC) remove the statistical redundancies. To reduce the blocking distortion, Loop-filter is applied to every decoded macroblock. This does not only improve the subjective visual quality of the reconstructed picture, but also improves the quality of the reference frames. As a result, the coding performance is also improved.

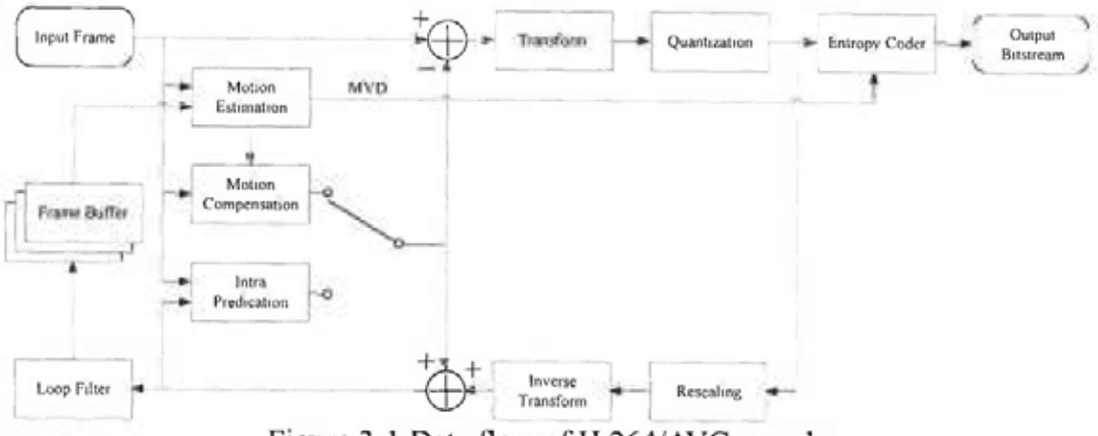


Figure 3-1 Data flow of H.264/AVC encoder.

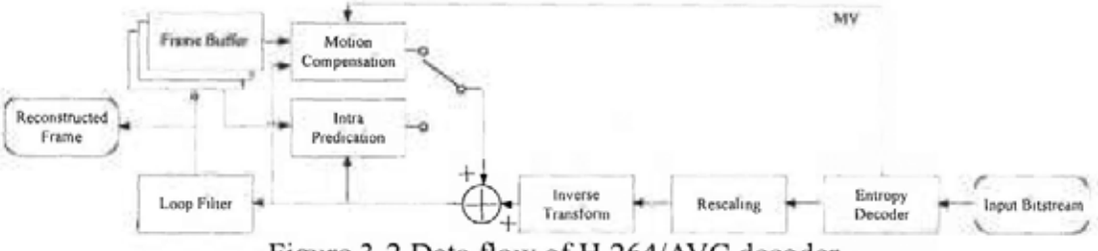


Figure 3-2 Data flow of H.264/AVC decoder.

In the standard, there are a number of profiles and levels which specify well-defined sets of syntax constraints for encoders and decoder processing capabilities for decoders. Profiles specify the syntax features while levels specify the parameters of these feature. There are a number of profiles defined and they target specific application areas. Some of these profiles are:

- **Baseline Profile:** Low-cost application such as video-conferencing and mobile applications.
- **Main Profile:** Standard-Definition (SD) TV broadcast.
- **Extended Profile:** Streaming video.
- **High Profile:** Video broadcast and storage, particularly for HDTV.
- **High 10 Profile:** Based on High Profile, it support up to 10 bit per sample.
- **High 4:2:2 Profile:** Based on High 10 Profile, it support 4:2:2: chroma sub-sampling format.

In this thesis, we focus on the High Profile which is particularly for HDTV. Depending on the application, different features are defined in different profiles. These features are commonly called coding tools. Numerous novel coding tools are adopted in H.264/AVC. Here are some of the typical coding tools defined in High Profile:

■ **Spatial Intra prediction:** A MB can be coded as one 16×16, four 8×8 or sixteen 4×4 sub-blocks. The pixel values of these blocks are predicted by their left and upper blocks (pixel A to M for 4×4 block and pixel A to Z for 8×8 block in Figure 3-3) There are 9 prediction modes (mode 0 to mode 8 in Figure 3-3) for 4×4 and 8×8 blocks while there are 4 prediction modes (mode 0 to mode 3 in Figure 3-4) for 16×16 block.

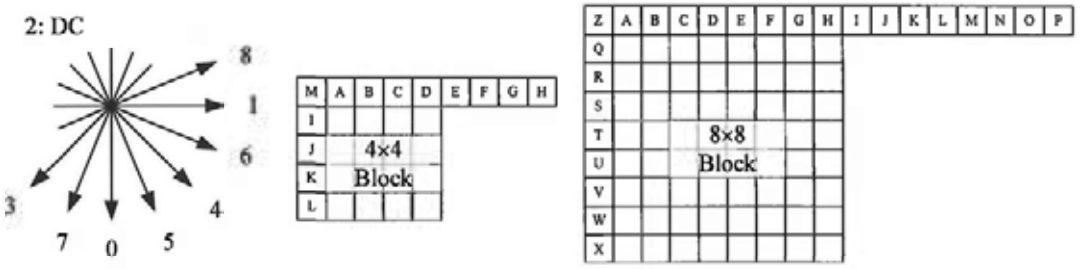


Figure 3-3 Nine intra prediction modes for 4×4 and 8×8 blocks.

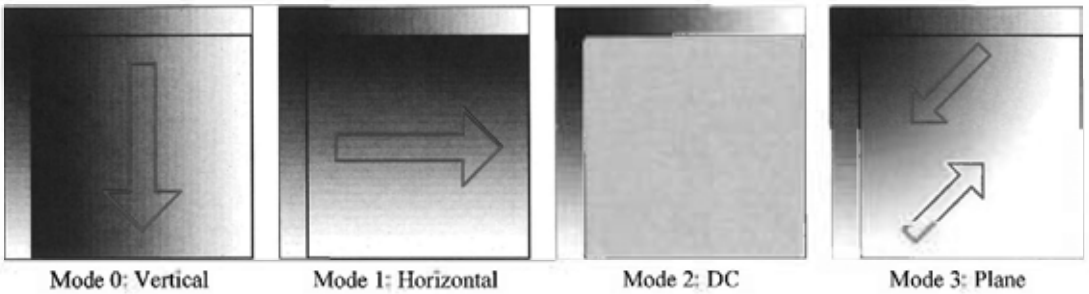


Figure 3-4 The intra prediction modes for 16×16 block.

■ **Multiple reference frames:** A maximum of 16 reference frames is supported. With more reference frames, more accurate prediction can be achieved. The predicted residue is reduced and hence the bit rate.

■ **Variable block-size motion compensation:** block-based motion compensation supports the block size from 4×4 up to 16×16. This allows a more precise segmentation of moving regions than fixed block-size motion compensation.

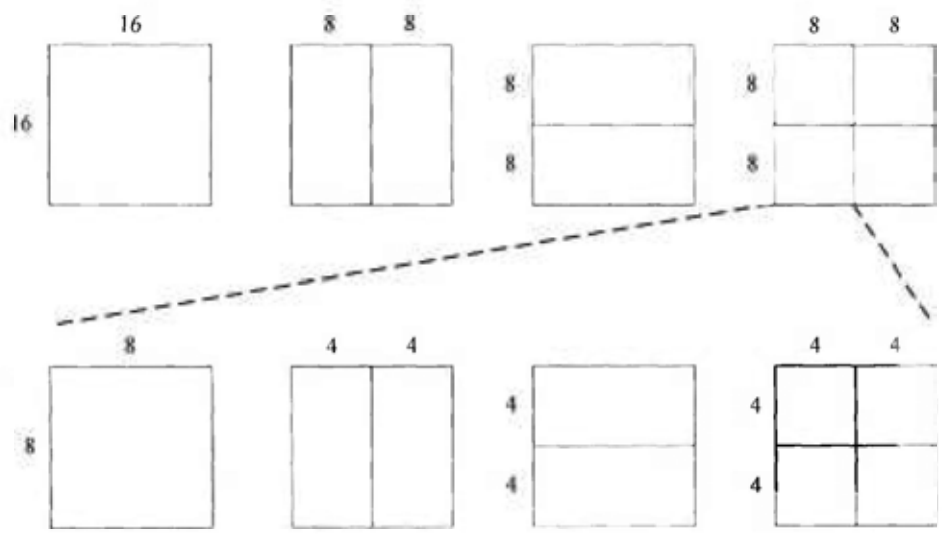


Figure 3-5 The variable block-size motion compensation partitions.

■ **Inter-prediction with sub-pixel accuracy:** Motion vectors may not fall exactly on the integer pixel grid. This is achieved by a 6-tap interpolation filter. A balance between the accuracy and the complexity, motion estimation in sub-pixel accuracy up to quarter pixel is supported in H.264/AVC. This reduces predicted residue and hence the bit rate.

■ **Order-4 and Order-8 ICT:** Predicted residues are transform with 4×4 ICT which produces less ringing artifacts in prior codec. In FRExt, 8×8 ICT is also allowed which provides a higher coding efficiency than 4×4 ICT in smooth regions.

The adaptive selection between the 4×4 and the 8×8 ICT improves the coding performance significantly.

- **Logarithmic step size quantization control:** This makes the rate management easier.

- **Loop filtering:** It is an in-loop low-pass filter which reduces the blocking artifacts common to block-based image compression techniques. It improves not only the visual quality but also the compression efficiency.

- **CABAC:** It is a very efficient entropy coder with better compression than most other encoding algorithms.

- **CAVLC:** It is a lower-complexity entropy coder than CABAC but with high coding efficiency also.

After a short review of H.264/AVC standard, the integration of order-16 transform to this standard will begin in the next section. From Section 3.2 to 3.6, our proposed implementation will be described. In Section 3.7, experimental result and analysis will be shown. Conclusions will be drawn in Section 3.8.

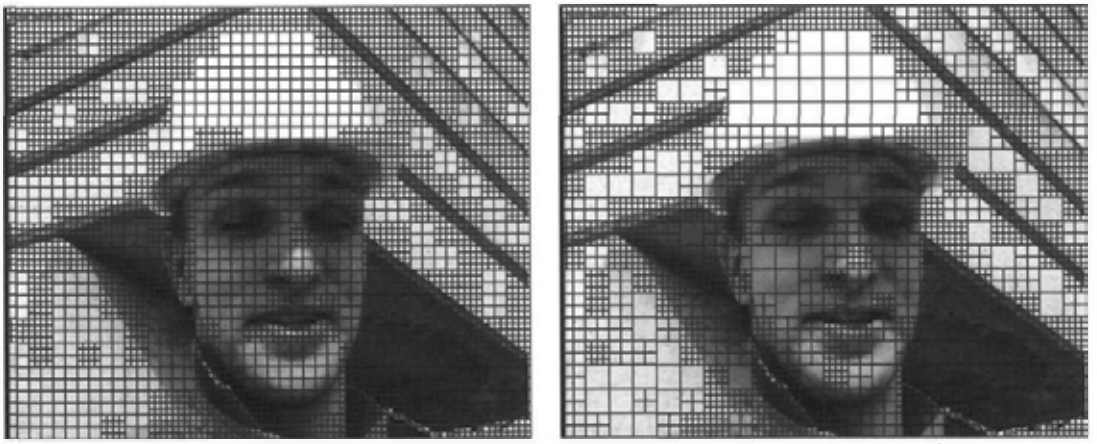
3.2 Transforms

In original standard, H.264/AVC specified an order-4 ICT for transforming the predicted residue. An order-8 ICT is added in FRExt. The integer kernel of order-4 and order-8 specified are:

$$\mathbf{E}_{IC}^{(4)} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \text{ and} \tag{3.1}$$

$$\mathbf{E}_{IC}^{(8)} = \begin{bmatrix} 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 \\ 12 & 10 & 6 & 3 & -3 & -6 & -10 & -12 \\ 8 & 4 & -4 & -8 & -8 & -4 & 4 & 8 \\ 10 & -3 & -12 & -6 & 6 & 12 & 3 & -10 \\ 8 & -8 & -8 & 8 & 8 & -8 & -8 & 8 \\ 6 & -12 & 3 & 10 & -10 & -3 & 12 & -6 \\ 4 & -8 & 8 & -4 & -4 & 8 & -8 & 4 \\ 3 & -6 & 10 & -12 & 12 & -10 & 6 & -3 \end{bmatrix}. \tag{3.2}$$

Here, the order-16 transforms described in the last chapter are integrated into the reference software of H.264/AVC. Together with order-4 and order-8 transform, an Arbitrary Block-size Transform (ABT) platform is formed. Only a single transform is selected in each MB in H.264/AVC. Different transforms can be used in different MB. It is decided in MB level. The proposed ABT platform keeps this remain unchanged. Either order-4, order-8 or order-16 transform is used in a single MB. An example is shown in Figure 3-6.



(a) Without 16×16 transform (b) With 16×16 transform

Figure 3-6 Arbitrary Block-size Transform in H.264/AVC.

The transforms being integrated are:

- Simple integer transform, $\mathbf{T}_{SI\text{-}H264}$,
- Simple integer transform, $\mathbf{T}_{SI\text{-}AVS}$,
- Modified ICT, MICT,
- Hybrid Integer Transform, $\mathbf{T}_{HI1}$,
- Hybrid Integer Transform, $\mathbf{T}_{HI2}$,
- Order-16 transform proposed by Wien, $\mathbf{T}_{Wien}$,
- Order-16 transform proposed by Lee et al., $\mathbf{T}_{Lee}$,
- Order-16 transform proposed by Joshi et al., $\mathbf{T}_{Joshi}$,
- LLMICT A-10, $\mathbf{T}_{LLM\text{-}A1}$,
- LLMICT B-1, $\mathbf{T}_{LLM\text{-}B1}$ and
- Modified CSFICT (MCSFICT), $\mathbf{T}_{MCSF}$.

They are tested individually in Section 3.7.

3.3 Quantization and Rescaling

The transform coefficients are quantized and rescaled before inverse transform. Since the norms of the integer transform basis vectors may not be the same. A scaling process is required to normalize the transform coefficients. In usual manner, this scaling process is embedded into the quantization and rescaling process. In this section, the method of quantization and rescaling with proposed transform in H.264 will be discussed.

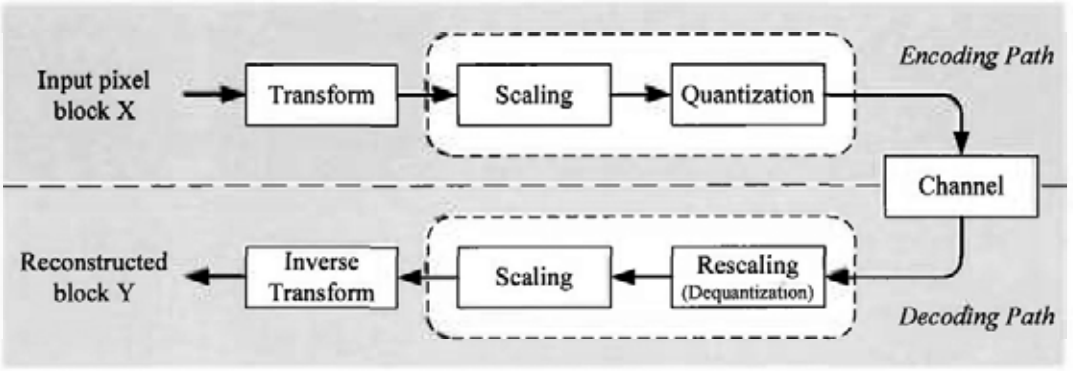


Figure 3-7 Data flow of quantization and rescaling in H.264/AVC.

3.3.1 Quantization

For simplicity, let us consider this process starting from DCT. Assume an $n \times n$ data patch $\mathbf{X}$ is transform with a DCT kernel $\mathbf{T}_{DCT}$ into an $n \times n$ coefficient patch $\mathbf{C}$. This process can be written as:

$$\mathbf{F}_{DCT} = \mathbf{T}_{DCT}\, \mathbf{X}\, \mathbf{T}_{DCT}^{T}. \tag{3.3}$$

When quantization with step size Q_{step} and offset Q_{offset} is applied to the coefficients, the coefficient (i, j) becomes:

$$F_q(i,j) = sign(F_{DCT}(i,j)) \times \left\lfloor \frac{F_{DCT}(i,j)}{Q_{step}} + Q_{offset} \right\rfloor . \tag{3.4}$$

Replace $\mathbf{T}_{DCT}$ by an ICT $\mathbf{T} = \mathbf{K}\,\mathbf{E}$ and recall that the 2-D scaling matrix $\mathbf{S}$ can replace the 1-D scaling matrix $\mathbf{K}$ described in last chapter. (3.4) becomes:

$$\mathbf{F}_{ICT} = \left(\mathbf{E}\,\mathbf{X}\,\mathbf{E}^T\right) \odot \mathbf{S} = \mathbf{G} \odot \mathbf{S} . \tag{3.5}$$

Note that $\odot$ is the element-to-element multiplication. Applying uniform quantization to integer transform coefficients with step size Q_{step} and rounding offset Q_{offset}:

$$\begin{aligned} F_q(i,j) &= sign(F_{ICT}(i,j)) \times \left\lfloor \frac{F_{ICT}(i,j)}{Q_{step}} + Q_{offset} \right\rfloor \\ &= sign(G(i,j)) \times \left\lfloor \frac{G(i,j) \times S(i,j)}{Q_{step}} + Q_{offset} \right\rfloor \\ &\approx sign(G(i,j)) \times \frac{\left\lfloor G(i,j) \times Quant(Q_{step};i,j) + Q_{offset} \cdot 2^{Qbits} \right\rfloor}{2^{Qbits}} \\ &= \left(sign(G(i,j)) \times \left\lfloor G(i,j) \times Quant(Q_{step};i,j) + Q_{offset} \cdot 2^{Qbits} \right\rfloor\right) >> Qbits. \end{aligned} \tag{3.6}$$

Quant is an integer approximation the quantization matrix:

$$Quant(Qstep;i,j) = round\left(\frac{\boldsymbol{S(i,j)}}{\boldsymbol{Q_{step}}} \cdot 2^{Qbits}\right). \tag{3.7}$$

In H.264, Q_{step} is defined as:

$$Q_{step} = 2^{\frac{QP-4}{6}} \tag{3.8}$$

where QP is the quantization parameter, integers from 0 to 51. **Quant** becomes:

$$
\begin{aligned}
Quant(Qstep;i,j) &= Quant(QP;i,j) \\
&= round\left(S(i,j) \times 2^{Qbits - \frac{QP-4}{6}} \right) \\
&\approx round\left(S(i,j) \times 2^{\frac{4-(QP\%6)}{6}} \cdot 2^{Qbits} \right) >> \left\lfloor \frac{QP}{6} \right\rfloor \\
&= QM(QP\%6;i,j) >> \left\lfloor \frac{QP}{6} \right\rfloor
\end{aligned}
\tag{3.9}
$$

Here **QM** is an integer matrix and the quantization can be implemented with integer only. It has a period of 6 such that:

$$QM(QP+6;i,j) = QM(QP;i,j). \tag{3.10}$$

In H.264, integer up to 64-bit accuracy is allowed. Assume that the input data are represented in b bits. The largest magnitude of $G(i, j)$ after 2-D transform is $2^b \times 256 * 256 = 2^{b+16}$. In High Profile, b is only 9. Integer with 64-bit accuracy is very sufficient to represent **G** without any fixed point error or overflow.

3.3.2 Rescaling

Similar to forward transform, a scaling process is required in inverse transform process. This scaling process is usually integrated into the rescaling process (also known as dequantization). The quantized coefficients $\mathbf{F}_q$ in the last section are rescaled.

$$\mathbf{F}_r = \mathbf{F}_q \odot \mathbf{S} \times Q_{step}. \tag{3.11}$$

It can be written as:

$$\begin{aligned} F_r(i,j) &= F_q(i,j) \times S(i,j) \times Q_{step} \\ &\approx F_q(i,j) \times \frac{round(S(i,j) \times Q_{step} \times 2^{DQbits})}{2^{DQbits}} \\ &\approx \left(F_q(i,j) \times dequant(Qstep;i,j)\right) >> DQbits \end{aligned} \tag{3.12}$$

dequant() can be expressed in terms of *QP*:

$$\begin{aligned} dequant(Qstep;i,j) &= round(S(i,j) \times Q_{step} \times 2^{DQbits}) \\ &= round(S(i,j) \times 2^{\frac{QP-4}{6}} \times 2^{DQbits}) \\ &\approx round(S(i,j) \times 2^{\frac{(QP\%6)-4}{6}} \times 2^{DQbits}) << \left\lfloor \frac{QP}{6} \right\rfloor \\ &= DQM(QP\%6;i,j) << \left\lfloor \frac{QP}{6} \right\rfloor \end{aligned} \tag{3.13}$$

Here **DQM** is the rescaling matrix in integer. $\mathbf{F}_r$ becomes

$$F_r(i,j) \approx \left(\left(F_q(i,j) \times DQM(QP;i,j)\right) << \left\lfloor \frac{QP}{6} \right\rfloor \right) >> (DQbits - 10) \tag{3.14}$$

Therefore, the reconstructed data patch **Y** is defined as:

$$\begin{aligned} \mathbf{Y} &= \left(\mathbf{E}^T \, \mathbf{F}_r \, \mathbf{E}\right) >> 10 \\ &= \left(\mathbf{E}^T \left(\left(F_q(i,j) \times DQM(QP;i,j)\right) << \left\lfloor \frac{QP}{6} \right\rfloor \right) \mathbf{E} \right) >> 10 \end{aligned} \tag{3.15}$$

Here *Qbits* and *DQbits* are transform dependent.

3.3.3 Example

Let us take an example to illustrate the quantization and the rescaling process in the proposed H.264/AVC platform. Suppose the LLMICT A1 is used ($\mathbf{E} = \mathbf{E}_{LLM\text{-}A1}$, $Qbits = 31$ and $DQbits = 29$) and QP = 28. According to (3.9), $\mathbf{QM}(QP\%6)$ is defined as:

$$QM(28\%6;i,j) = round\left(S(i,j) \times 2^{\frac{4-(28\%6)}{6}} \cdot 2^{31} \right)$$

$$\mathbf{QM}(28\%6) = \begin{bmatrix} 2048 & 2135 & 2204 & 2242 & 2151 & 2242 & 2204 & \cdots & 2204 & 2135 \\ 2135 & 2225 & 2298 & 2337 & 2242 & 2337 & 2298 & \cdots & 2298 & 2225 \\ 2204 & 2298 & 2372 & 2413 & 2315 & 2413 & 2372 & & 2372 & 2298 \\ 2242 & 2337 & 2413 & 2455 & 2356 & 2455 & 2413 & & 2413 & 2337 \\ 2151 & 2242 & 2315 & 2356 & 2260 & 2356 & 2315 & & 2315 & 2242 \\ 2242 & 2337 & 2413 & 2455 & 2356 & 2455 & 2413 & & 2413 & 2337 \\ 2204 & 2298 & 2372 & 2455 & 2315 & 2413 & 2372 & & 2372 & 2298 \\ \vdots & \vdots & & & & & & \ddots & \vdots & \vdots \\ 2204 & 2298 & 2372 & 2413 & 2315 & 2413 & 2372 & \cdots & 2372 & 2298 \\ 2135 & 2225 & 2298 & 2337 & 2242 & 2337 & 2298 & \cdots & 2298 & 2225 \end{bmatrix} \tag{3.16}$$

Consider an intra-predicted block $\mathbf{X}$:

-2	-1	9	26	43	52	53	51	50	50	50	51	53	57	55	52
-3	10	36	42	40	43	42	41	39	37	39	40	40	44	45	42
16	24	19	-1	11	28	24	22	21	20	22	21	25	29	30	28
6	7	10	12	12	4	1	5	0	-2	6	11	10	11	13	12
-1	2	3	1	2	2	0	1	2	2	2	0	4	4	1	-4
-2	0	2	-3	-3	0	0	1	0	4	4	4	6	4	0	-10
0	0	-15	-8	3	-1	2	1	1	4	2	5	4	1	-8	-26
6	-31	-54	-10	3	-6	-3	1	-2	-4	2	8	4	-3	-1	0
-42	-53	-9	1	-5	-1	-1	-1	0	1	2	4	2	-4	-3	0
4	0	-1	-2	-4	-7	-4	1	0	-1	1	3	0	-3	2	6
-3	-3	-1	-4	-3	-2	-1	3	4	2	-14	-10	1	-2	0	1
1	0	1	-2	-1	1	1	-1	-3	-5	-11	-3	1	-2	-1	-2
1	-2	-2	-4	-4	-3	0	1	0	3	6	-1	-8	1	3	-1
1	1	-1	-3	-3	-1	0	0	1	4	3	-1	-2	2	3	0
3	2	-1	-3	-2	0	0	1	4	5	3	0	-1	-1	0	-3
1	-2	-3	-1	0	3	4	3	3	3	1	-3	-4	-5	-8	-4

Using (3.6), the quantized transform coefficients $\mathbf{F}_q$ are:

6	-3	-2	-1	-1	1	0	1	0	1	0	1	0	1	0	0
9	-3	-1	-2	-1	1	0	0	0	0	-1	0	0	0	0	0
9	-1	0	-1	1	-1	-1	-1	-1	-1	-1	0	1	0	0	0
5	-2	-1	-2	0	-1	-1	-1	0	0	1	1	0	1	0	0
3	-3	-3	-1	-1	0	1	1	1	1	1	0	0	0	0	0
2	-1	-2	1	-1	1	1	1	0	1	-1	0	-1	-1	0	0
1	1	-1	0	0	1	-1	0	-1	-1	0	0	-1	0	0	0
0	-1	-1	-1	-1	-1	-1	-1	0	1	1	0	1	0	0	0
-2	-1	-1	-1	-1	0	1	1	1	1	1	0	0	0	0	0
-1	0	1	1	1	1	1	1	1	0	-1	-1	-1	-1	0	0
1	1	1	1	1	1	1	0	-1	0	-1	-1	0	0	0	0
0	-1	0	0	-1	0	-1	-1	-1	-1	0	1	1	1	0	0
-1	-1	-1	-1	1	1	1	0	0	-1	1	0	0	0	0	0
0	-1	1	1	2	1	1	1	0	0	-1	-1	-1	-1	-1	0
0	1	0	1	1	1	1	0	-1	-1	0	0	0	0	0	0
0	-1	-1	-1	-1	-1	-1	0	-1	-1	0	1	1	1	1	0

According to (3.13), **DQM** is

$$DQM(QP\%6;i,j) = round(S(i,j)\times 2^{\frac{(28\%6)-4}{6}}\times 2^{DQbits})$$

$$\mathbf{DQM}(28\%6) = \begin{bmatrix} 512 & 534 & 551 & 561 & 538 & 561 & \cdots & 551 & 534 \\ 534 & 556 & 574 & 584 & 561 & 584 & \cdots & 574 & 556 \\ 551 & 574 & 593 & 603 & 579 & 603 & & 593 & 574 \\ 561 & 584 & 603 & 614 & 589 & 514 & & 603 & 584 \\ 538 & 561 & 579 & 589 & 565 & 589 & & 579 & 561 \\ 561 & 584 & 603 & 614 & 589 & 614 & & 603 & 584 \\ \vdots & \vdots & & & & & \ddots & \vdots & \vdots \\ 551 & 574 & 593 & 603 & 579 & 603 & \cdots & 593 & 574 \\ 534 & 556 & 574 & 584 & 561 & 584 & \cdots & 574 & 556 \end{bmatrix} \tag{3.17}$$

Using (3.14) and (3.15), the reconstructed pixels, **Y**, are:

-10	-9	6	24	44	60	65	57	52	50	54	57	59	61	64	48
-6	-3	45	52	43	40	41	50	47	36	37	47	42	43	57	42
16	28	21	-19	6	38	16	21	26	17	15	16	27	28	42	43
12	9	12	10	15	-1	-7	9	-7	-16	3	7	5	13	15	14
-5	-5	-5	1	5	3	2	-2	0	-1	-3	-3	-1	7	0	-11
-8	-1	-8	-3	-3	9	13	4	-1	-2	2	9	15	7	3	-19
14	11	-13	-17	16	7	-2	-6	-3	7	10	21	1	8	3	-36
24	-26	-92	-2	14	-19	0	-7	-7	-2	10	9	0	-3	2	-4
-51	-97	15	14	-3	10	1	-5	-10	6	-1	3	15	-1	-5	0
12	6	5	6	-10	-7	5	10	-8	0	7	-7	4	-2	7	1
-8	-9	0	-6	-3	-4	5	1	3	6	-25	-12	3	-6	2	1
5	-1	8	-3	-10	2	0	-8	-4	1	-16	-1	6	-13	-1	-18
0	-5	2	4	0	-4	-5	1	-1	3	11	3	-7	4	16	-5
11	0	3	-1	-5	2	7	4	1	3	11	3	-9	-3	-9	1
-6	2	10	3	-7	-10	-17	3	13	1	8	6	3	10	11	1
12	0	-3	-7	4	11	6	2	2	12	0	-12	-3	-3	-9	-4

3.4 Syntax Structure

3.4.1 New Syntax Elements

In the proposed platform, two new syntax elements, *I16Flag* and *CBP16*, are introduced. *I16Flag* is a flag indicating the transform size used in the current MB. The bit patterns of *I16Flag* are as shown in Table 3-1. Notice that 16×8 and 8×16 transforms are not used in intra-block.

One may suggest that the transform size sticks to the ME partition size. This saves the overhead indicating the transform size. However, we found that the scheme has a lower coding performance than the one with overhead. This is simply because the best transform cannot be selected when the transform size links to the ME partition size.

Transform Size	*I16Flag*	
	Intra-block	Inter-block
4×4 or 8×8	0	0
16×16	1	10
16×8	---	110
8×16	---	111

Table 3-1 Bit patterns for *I16flag*

Another new syntax element is *CBP16*. It is similar to *CBP* in existing standard but it is specific for larger transform size. It indicates the present of non-zero luma transform coefficients in the sub-blocks when larger transform is used. If 16×16 transform is used, its length is 1 bit. If 16×8 or 8×16 transform is used, it is 2-bit long.

3.4.2 Intra Block Syntax Structure

When order-16 transform is introduced, the syntax structure has to be changed. The change of syntax structure in intra block is relatively simpler. In H.264/AVC, 4×4, 8×8 and 16×16 Intra predications are allowed. *MB Type* indicates which predication is used as shown in Figure 3-8. Since the transform size must not be bigger than the intra prediction partition size, 16×16 transform is allowed only when 16×16 intra prediction is used. Originally 4×4 transform is used when the MB is 16x16 intra predicted. To distinguish the use of 4×4 and 16×16 transform, *I16flag* is added to the bit stream. When this flag is "1", 16×16 transform is used. Otherwise, 4×4 transform is used. Other predictions remain the same as the standard.

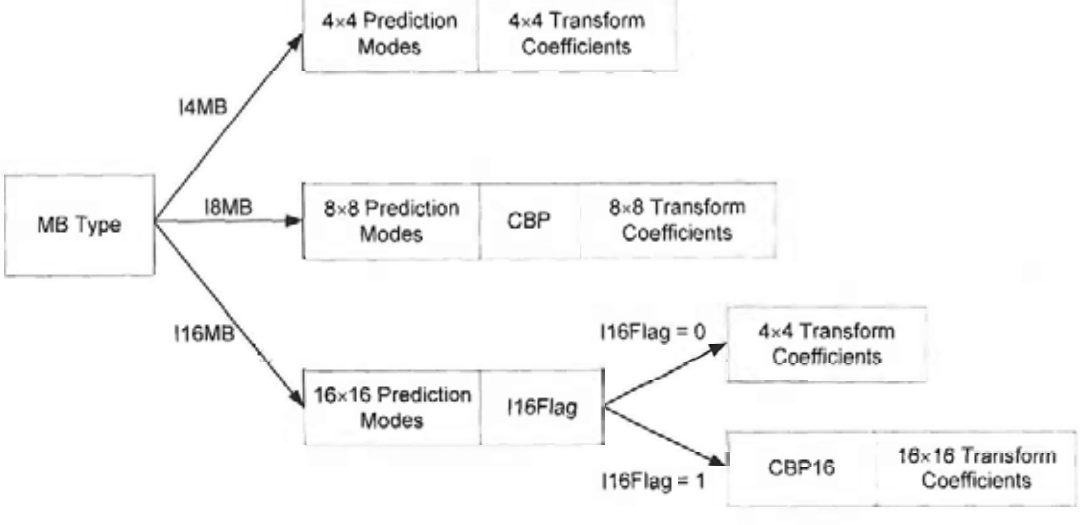


Figure 3-8 Syntax structure for intra-block

3.4.3 Inter Block Syntax Structure

The inter block syntax is slightly more complex. 16×16, 16×8 and 8×16 are available when larger inter-predicted partition is used. When the MB is predicted in direct mode or the prediction partition size is 16×16, 16×8 or 8×16, *I16flag* is inserted

into the bit stream to indicate the transform size as shown in Figure 3-9. The mapping between *I16flag* and the transform size is shown in Table 3-1. If larger transform size is not used, it follows the original decoding routine with smaller transform size.

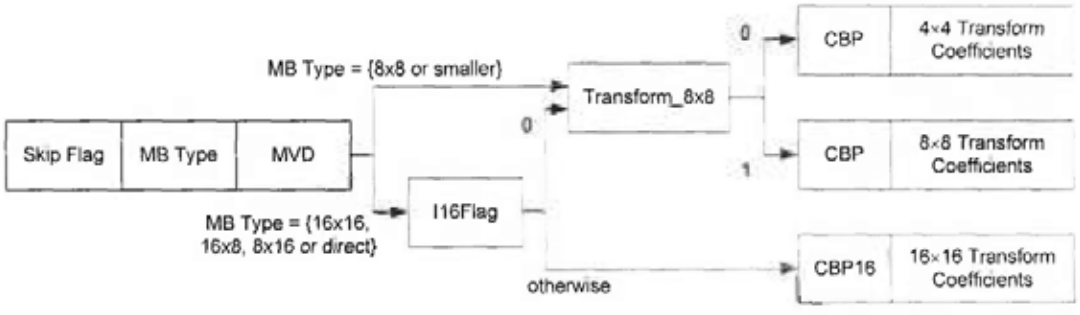


Figure 3-9 Syntax structure for inter-block

3.5 Entropy Coding

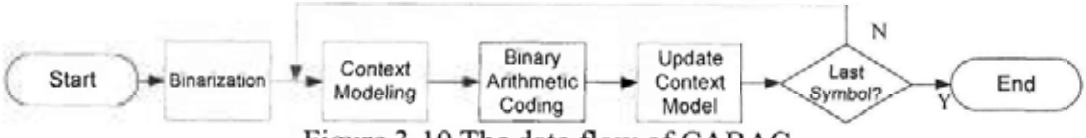


Figure 3-10 The data flow of CABAC.

The transform coefficients and some other syntax elements are coded with an entropy coder. The entropy coding in our implementation is Context-Adaptive Binary Arithmetic Coding (CABAC). This is because it has a higher efficiency than CAVLC. In CABAC, different syntax elements reference different context model. Each context model contains the most probable symbol (MPS) of this context and the probability (or the state) of this MPS. The flow of a CABAC coder is shown in Figure 3-10. In binarization, the input syntax elements are first changed into binary symbols, '0' or '1'. These binary symbols are called "bins". These bins are mapped to different context models by context indexing. The binary arithmetic coding (BAC) engine in CABAC obtains the MPS and the probability of the context model to encode the bin. Bit stream will be generated in this coding engine according to the status of the context model (MPS and its probability) and the incoming bins. The status of the corresponding context model will be updated after encoding every bin. This ensures the status of the context model that follows the statistics of the encoded symbols. As a result, this coder is adaptive to the context. In our implementation, the coding engine is the one specified in H.264/AVC. The context model indexing for newly introduced syntax elements, such as *I16flag* and *CBP16*, are added.

As stated in the last section, *I16flag* is a flag indicating the use of larger transform. The simplest way is to use a single context model. This will capture the global statistics of this flag within the same frame. However, this fails to capture its local statistics. It is expected that MB coded with similar transform size are located in group closely. Single context model is not able to react quickly when this flag changes. As a result, the context index of the *I16flag* is according to the neighboring MB in our proposed platform. Two previously coded MB, the upper MB and the left MB, are referenced. The indexing method is shown in Table 3-2. The index will be assigned according to the value of *I16flag* of the neighbor. Three context models are required. This will not increase the complexity significantly but allow the context modeling to capture the local and also the global statistics of *I16flag*. Although it is not too much, a bit rate reduction of less than 1% is observed with this context modeling.

The context model of the flag CBP16, which indicates the presence of non-zero order-16 transform coefficients, is also indexed with similar method as I16flag. The index depends on the presence of non-zero coefficient (of any transform size) in the neighboring MBs. The indexing of CBP16 is shown in Table 3-3.

Current block *I16flag* context model index.		*I16flag* in Upper MB	
		1	0
I16flag in Left MB	1	2	1
	0	1	0

Table 3-2 Context model index of *I16flag*

Current block *CBP16* context model index		Non-zero coefficients in Upper MB	
		Present	Absent
Non-zero coefficients in Left MB	Present	3	1
	Absent	2	0

Table 3-3 Context model index of *CBP16*.

3.6 Rate-Distortion Optimization

Rate-Distortion Optimization (RDO) is always enabled in our experiment. The Lagrangian RD Optimizer adopted in the H.264/AVC reference software is used. All possible combinations (prediction modes and transform sizes) of each MB are tested. For each combination, the bit rate (R) and the distortion (D) in sum of squared different (SSD) are measured. The RD cost (J) of each combination is calculated as:

$$J = D + \lambda R . \tag{3.18}$$

λ is the lagrangian multiplier which depends on the QP value. The combination offering the smallest RD cost is chosen.

3.7 Experiment and Analysis

The analysis here is on the H.264/AVC reference software JM16.2 [12]. Those order-16 transforms mentioned in Section 3.2 are integrated into the software platform. It is implemented as the described in previous sections. The testing conditions are based those stated in VCEG-AJ10 [13]. Detailed testing conditions are listed in Table 3-4. The resolutions of the video sequences in our test are ranged from CIF (352×288) to HD (1920×1080). We use the hierarchical-B prediction structure which provides the best coding performance among different prediction structures. The analysis will be in several aspects. First, the objective RD performance will be analyzed. The BD-bit rate and the BD-PSNR mentioned in [14] are computed and compared (See Chapter 1 for more detail). RD curves will also be shown. Second, subjective evaluation will be given. The decoded pictures will be shown. These pictures will demonstrate the differences among different transforms. Third, the usage of order-16 transform will be investigated. The conditions that order-16 transform is chosen will be discussed. Lastly, the conditions that may lower the performance of the ABT with order-16 transform will be pointed out.

Platform	JM16.2
Anchor	JM16.2 (Order-8 and -4 transform enabled. No order-16 transform)
Prediction Structure	Hierarchical B Structure with 7 B-frames IbBbBbBbP…
Hierarchical Coding Settings	HierarchicalCoding = 2, ReferenceRecoder = 1, PocMemoryManagement = 1, HierarchyLevelQPEnable = 1
Intra Period	Only the first frame is intra predicted
Frame encoded	More than 120 frames: CIF, WQVGA or WVGA: 297 frames, 720p: 145 frames, 1080p 121 frames.
Number of Reference Frames	4
Entropy Coding	CABAC
Transform8x8Mode	Enable
Scaling Matrix Present	Disable
Rate Control	Disable
RDO	Enable
RDO_Q	Disable
Loop Filter	Enable
Subpel Motion Estimation	1/4 pixel enabled
Motion Estimation Search Range	64
Motion Estimation Method	Enhanced Predictive Zonal Search
QP = {QP_I, QP_P, QP_B}	{22, 23, 24}, {27, 28, 29}, {32, 33, 34}, {37, 38, 39}

Table 3-4 Testing conditions in H.264/AVC platform

3.7.1 RD analysis (Objective Evaluation)

The BD bit rate and the equivalent BD PSNR are shown in Table 3-5 and Table 3-6 respectively. It is shown that the addition of order-16 transform significantly improves the coding performance. In all the tested cases, the bit rate reductions are observed. From Table 3-5, we can see that there is a slowly increasing trend in the average bit rate reduction when the resolution of the video frame is increasing. Of course, the bit rate reduction depends on the nature of the individual video sequence more rather than the frame resolution. This will be discussed in later section in this chapter. It is also observed that a maximum of 12.11% bit rate reduction (equivalent to 0.47 dB) is obtained when the order-16 transform $\mathbf{T}_{Joshi}$ is used to encode 1080p sequence "Sunflower". In [15], it is reported that this transform offers a bit rate reduction around 36% together with Mode-Dependent Directional Transform (MDDT) [16] and bigger motion partition up to 32×32. In our experiment, purely 16×16 transform is added. MDDT and the bigger partition size are not included.

On average, $\mathbf{T}_{joshi}$ gives the largest gain. An overall bit rate reduction of 6.21% (0.24 dB equivalent) is observed in all test sequences. Not very far, it is followed by LLMICT-A1, MCSFICT and LLMICT-B1 which are only lagged behind by 0.03%, 0.08% and 0.24% respectively. The differences are so small but the three proposed transforms are simpler. They save around 10% computation time with respect to $\mathbf{T}_{joshi}$ shown in last chapter. $\mathbf{T}_{Lee}$ is lagged by 0.51% (or 0.02 dB equivalent) but it is requires more computation than $\mathbf{T}_{joshi}$. MICT, $\mathbf{T}_{HI1}$ and $\mathbf{T}_{HI2}$ are developed with the same method. Their performances are very similar. Their bit rate reductions are 5.65%, 5.51% and 5.56% respectively. $\mathbf{T}_{Wien}$ also shows a bit rate reduction of 5.05%.

$\mathbf{T}_{S1\text{-}H264}$ and $\mathbf{T}_{S1\text{-}AVS}$ are the two simplest transforms in our test. Lower coding performance is expected. However, they also offer average bit rate reductions of 4.83% and 4.16% respectively. They are lagged behind by $\mathbf{T}_{joshi}$ by 1.38% and 2.05%. But they save over 20% computation time comparing with $\mathbf{T}_{joshi}$.

From the RD-curves shown in Figure 3-11 and Figure 3-12, the coding performance differences among different transforms are small. It is not easy to distinguish. However, the figures clearly show that the tested RD curves are almost parallel with the anchor RD curve. This implies that the gain is not only obtained at the low bit rates but throughout the tested QP range.

		$T_{SI\text{-}H264}$	$T_{SI\text{-}AVS}$	MICT	T_{HI1}	T_{HI2}	T_{Wien}	T_{Lee}	T_{Joshi}	$T_{LLM\text{-}A1}$	$T_{LLM\text{-}B1}$	T_{MCSF}
CIF (352×288)	Foreman	-3.61	-2.61	-4.22	-3.76	-3.67	-3.88	-4.25	-4.45	**-4.78**	-3.94	-4.50
	Mobile	-5.69	-2.42	-5.69	-5.91	-5.97	-5.74	-5.86	**-6.13**	-5.98	**-6.13**	-5.95
	News	-5.59	-3.43	-6.00	-5.44	-5.74	-5.80	-6.07	-5.99	**-6.22**	-6.11	-5.97
	Paris	-6.20	-3.43	-6.21	-6.02	-6.12	-6.30	**-6.34**	-5.58	-6.29	-6.20	-6.12
	Tempete	-3.32	-2.42	-3.85	-2.71	-2.63	-3.40	-4.03	-3.09	**-4.30**	-3.09	-4.26
	Average	-4.88	-2.86	-5.19	-4.77	-4.83	-5.02	-5.31	-5.05	**-5.51**	-5.09	-5.39
WQVGA (416×240)	BasketballPass	-5.14	-3.35	-5.49	-5.48	-5.46	-4.88	-5.40	-5.57	**-5.59**	-5.46	-5.50
	BlowingBubble	-7.17	-4.45	-7.60	-7.58	-7.61	-7.11	-7.74	**-7.97**	-7.90	-7.87	-7.85
	BQSquare	-3.50	-4.91	-5.11	-5.08	-4.99	-5.06	-5.25	**-5.33**	-5.27	-5.42	-5.22
	Flowervase	-5.61	-6.42	-6.47	-6.31	-6.65	-6.11	-6.23	**-6.78**	-6.40	-6.36	-6.50
	Average	-5.36	-4.78	-6.17	-6.11	-6.18	-5.79	-6.16	**-6.41**	-6.29	-6.28	-6.27
WVGA (832×480)	BasketballPass	-6.08	-8.40	-9.20	-9.18	-9.13	-8.63	-9.67	**-9.96**	-9.91	-9.86	-9.88
	BQMall	-7.08	-5.22	-7.57	-7.43	-7.48	-7.05	-7.73	**-8.02**	-7.99	-7.96	-7.97
	PartyScene	-8.87	-4.94	-9.10	-9.08	-9.07	-8.81	-9.19	-9.22	-9.25	-9.21	**-9.27**
	RaceHorses	-2.19	-1.89	-2.55	-2.52	-2.57	-2.13	-2.77	-3.07	**-3.08**	-2.93	**-3.08**
	Mobisode2	-1.89	-3.92	-3.86	-4.34	-4.58	-3.62	-4.26	**-4.89**	-4.23	-4.64	-4.33
	Average	-5.22	-4.87	-6.46	-6.51	-6.57	-6.05	-6.72	**-7.03**	-6.89	-6.92	-6.91

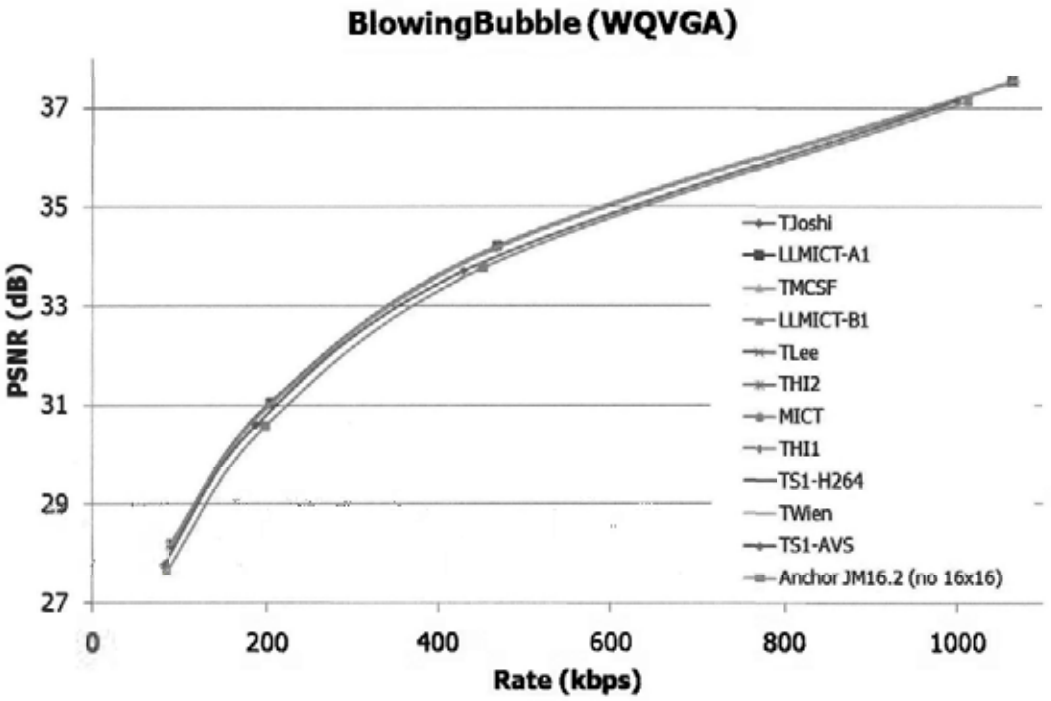
720p (1280×720)	Bigship	-3.30	-2.01	-3.89	-3.81	-3.95	-3.16	-3.91	-4.31	-3.80	-4.22	**-4.32**
	City	-2.24	-2.56	-3.29	-3.55	-3.79	-3.06	-3.76	**-4.60**	-4.19	-4.42	-4.15
	Crew	-7.36	-7.28	**-7.95**	-6.39	-6.68	-7.19	-7.83	-6.79	-7.65	-6.51	-7.82
	Night	-3.29	-2.24	-3.62	-3.60	-3.69	-3.42	-3.84	**-4.18**	-3.92	-4.08	-3.95
	Raven	-3.84	-3.83	-4.66	-4.32	-4.13	-4.39	-4.87	-5.12	-5.40	-4.61	**-5.77**
	Parkrun	-3.20	-2.41	-3.81	-3.80	-3.81	-3.71	-4.68	**-5.27**	-5.22	-4.87	-5.17
	ShuttleStart	-3.59	-3.69	-4.20	-4.10	-3.94	-3.44	-3.92	**-4.40**	-4.30	-4.20	-4.20
	Average	-3.83	-3.43	-4.49	-4.22	-4.28	-4.05	-4.69	-4.95	-4.93	-4.70	**-5.05**
1080p (1920×1080)	Cactus	-4.71	-4.56	-5.26	-5.41	-5.50	-4.45	-5.39	**-6.35**	-6.00	-6.12	-5.96
	Sunflower	-10.02	-9.33	-10.40	-10.90	-11.58	-10.40	-9.87	**-12.11**	-11.28	-11.70	-11.19
	BasketballDrive	-6.82	-6.39	-7.63	-7.42	-7.17	-6.47	-7.46	-7.77	-7.72	-7.54	**-7.89**
	Kimono1	-4.90	-4.72	-6.41	-6.26	-6.15	-4.15	-6.09	**-8.36**	-8.15	-7.32	-7.73
	Pedestrian	-7.10	-5.88	-7.85	-8.18	-7.96	-6.86	-7.95	**-8.82**	-8.77	-8.57	-8.56
	RiverBed	-1.82	-1.59	-4.21	-4.01	-3.97	-1.33	-4.07	**-7.10**	-6.70	-5.70	-6.36
	Station2	-3.44	-3.31	-4.33	-4.11	-4.19	-3.74	-4.41	**-5.09**	-4.98	-4.71	-4.85
	RushHour	-2.60	-3.08	-3.30	-3.05	-3.20	-2.02	-2.92	**-3.86**	-3.85	-3.50	-3.52
	Average	-5.18	-4.86	-6.17	-6.17	-6.22	-4.93	-6.02	**-7.43**	-7.18	-6.90	-7.01
Average		-4.83	-4.16	-5.65	-5.51	-5.56	-5.05	-5.72	**-6.21**	-6.18	-5.97	-6.13

Table 3-5 Experimental Results of different transforms in H.264/AVC platform (BD-bitrate, %)

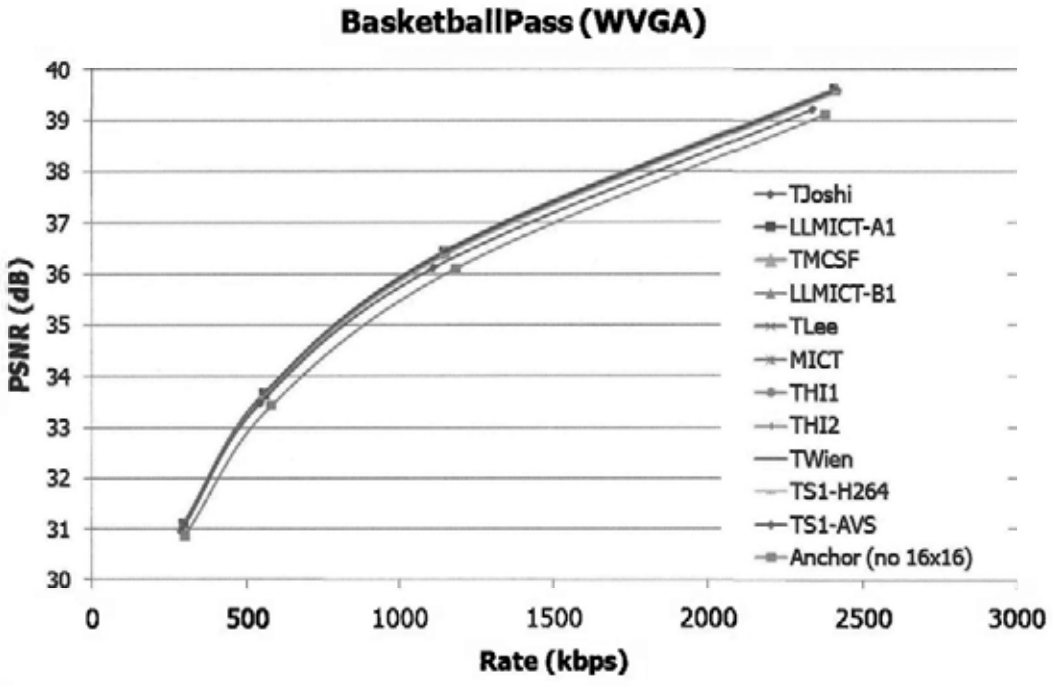
		$T_{S1\text{-}H264}$	$T_{S1\text{-}AVS}$	MICT	T_{HI1}	T_{HI2}	T_{Wen}	T_{Lee}	T_{Ioshi}	$T_{LLM\text{-}A1}$	$T_{LLM\text{-}B1}$	T_{MCSF}
CIF (352×288)	Foreman	0.16	0.11	0.18	0.16	0.16	0.17	0.19	0.19	**0.21**	0.17	0.20
	Mobile	0.25	0.10	0.25	0.26	0.26	0.25	0.26	**0.27**	0.26	**0.27**	0.26
	News	0.29	0.18	0.32	0.29	0.30	0.31	0.32	0.32	**0.33**	0.32	0.31
	Paris	0.32	0.18	0.32	0.31	0.32	**0.33**	**0.33**	0.29	**0.33**	0.32	0.32
	Tempete	0.13	0.09	0.15	0.11	0.10	0.13	0.16	0.12	**0.17**	0.12	**0.17**
	Average	0.23	0.13	0.24	0.23	0.23	0.24	0.25	0.24	**0.26**	0.24	0.25
WQVGA (416×240)	BasketballPass	0.25	0.16	0.26	0.26	0.26	0.23	0.26	**0.27**	**0.27**	0.26	0.26
	BlowingBubble	0.28	0.17	0.30	0.30	0.30	0.28	0.31	**0.32**	0.31	0.31	0.31
	BQSquare	0.17	0.12	0.18	0.18	0.18	0.18	**0.19**	**0.19**	**0.19**	**0.19**	0.18
	Flowervase	0.32	0.28	0.32	0.32	0.33	0.31	0.31	**0.34**	0.32	0.32	0.33
	Average	0.26	0.18	0.27	0.27	0.27	0.25	0.27	**0.28**	0.27	0.27	0.27
WVGA (832×480)	BasketballPass	0.35	0.25	0.38	0.38	0.38	0.36	0.40	**0.42**	0.41	0.41	0.41
	BQMall	0.30	0.22	0.32	0.31	0.32	0.30	0.33	**0.34**	**0.34**	**0.34**	**0.34**
	PartyScene	0.39	0.21	0.40	0.40	0.40	0.39	0.40	0.40	**0.41**	0.40	**0.41**
	RaceHorses	0.09	0.08	0.10	0.10	0.10	0.09	0.11	**0.12**	**0.12**	**0.12**	**0.12**
	Mobisode2	0.04	0.12	0.12	0.14	0.14	0.12	0.13	**0.15**	0.13	**0.15**	0.13
	Average	0.23	0.18	0.26	0.27	0.27	0.25	0.27	**0.29**	0.28	0.28	0.28

720p (1280×720)	Bigship	0.08	0.05	0.10	0.10	0.10	0.08	0.10	**0.11**	0.10	**0.11**	**0.11**
	City	0.06	0.08	0.09	0.10	0.10	0.08	0.10	**0.13**	0.12	0.12	0.12
	Crew	0.19	0.19	0.20	0.16	0.17	0.18	**0.20**	0.17	0.19	0.17	**0.20**
	Night	0.11	0.07	0.12	0.12	0.12	0.12	0.13	**0.14**	0.13	**0.14**	0.13
	Raven	0.14	0.14	0.18	0.16	0.15	0.17	0.18	0.19	0.20	0.17	**0.22**
	Parkrun	0.12	0.09	0.15	0.15	0.15	0.14	0.18	**0.20**	**0.20**	0.19	**0.20**
	ShuttleStart	0.11	0.11	0.13	0.13	0.12	0.11	0.12	**0.14**	0.13	0.13	0.13
	Average	0.12	0.10	0.14	0.13	0.13	0.13	0.14	0.15	0.15	0.15	**0.16**
1080p (1920×1080)	Cactus	0.14	0.12	0.15	0.16	0.16	0.13	0.15	**0.18**	0.17	0.17	0.17
	Sunflower	0.39	0.35	0.41	0.43	0.45	0.39	0.39	**0.47**	0.43	0.46	0.43
	BasketballDrive	0.20	0.18	0.22	0.22	0.21	0.19	0.21	**0.23**	**0.23**	0.22	**0.23**
	Kimono1	0.19	0.18	0.26	0.25	0.25	0.16	0.24	**0.34**	0.33	0.29	0.31
	Pedestrian	0.28	0.23	0.31	0.32	0.31	0.27	0.31	**0.35**	**0.35**	0.34	0.34
	RiverBed	0.08	0.07	0.19	0.18	0.18	0.06	0.18	**0.32**	0.31	0.26	0.29
	Station2	0.17	0.14	0.22	0.21	0.21	0.19	0.22	**0.26**	0.25	0.24	0.24
	RushHour	0.08	0.09	0.10	0.09	0.10	0.06	0.09	**0.12**	**0.12**	0.11	0.11
	Average	0.19	0.17	0.23	0.23	0.23	0.18	0.22	**0.28**	0.27	0.26	0.27
Average		0.20	0.15	0.22	0.22	0.22	0.20	0.22	**0.24**	**0.24**	**0.24**	**0.24**

Table 3-6 Experimental Results of different transforms in H.264/AVC platform (BD-PSNR, dB)

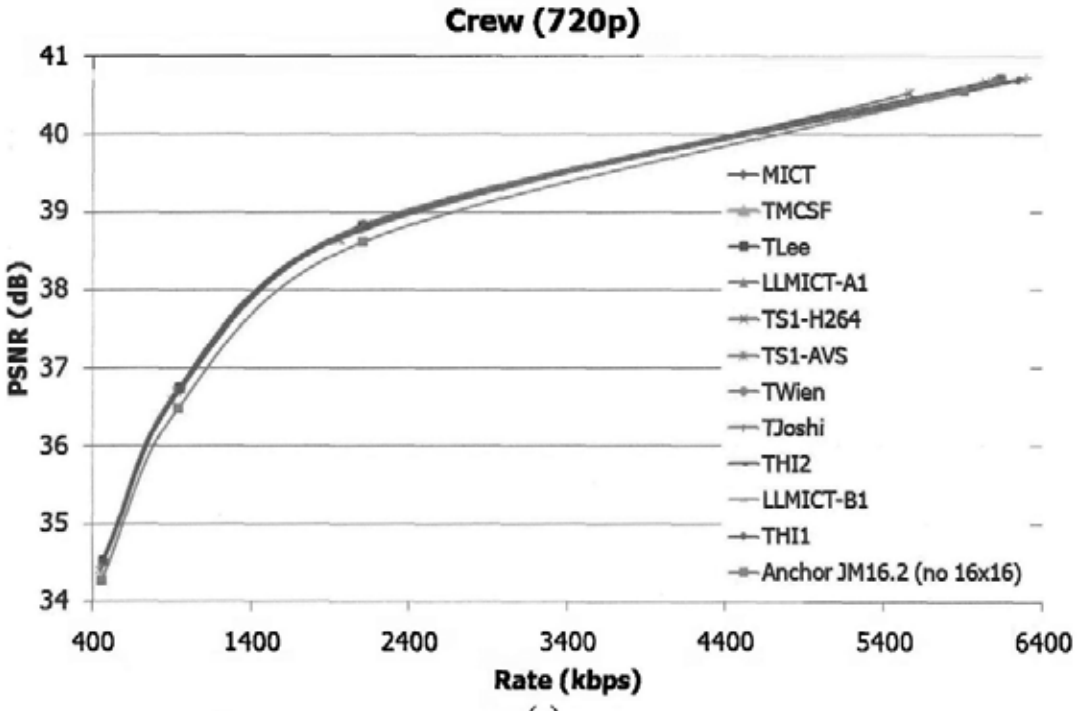


(a)

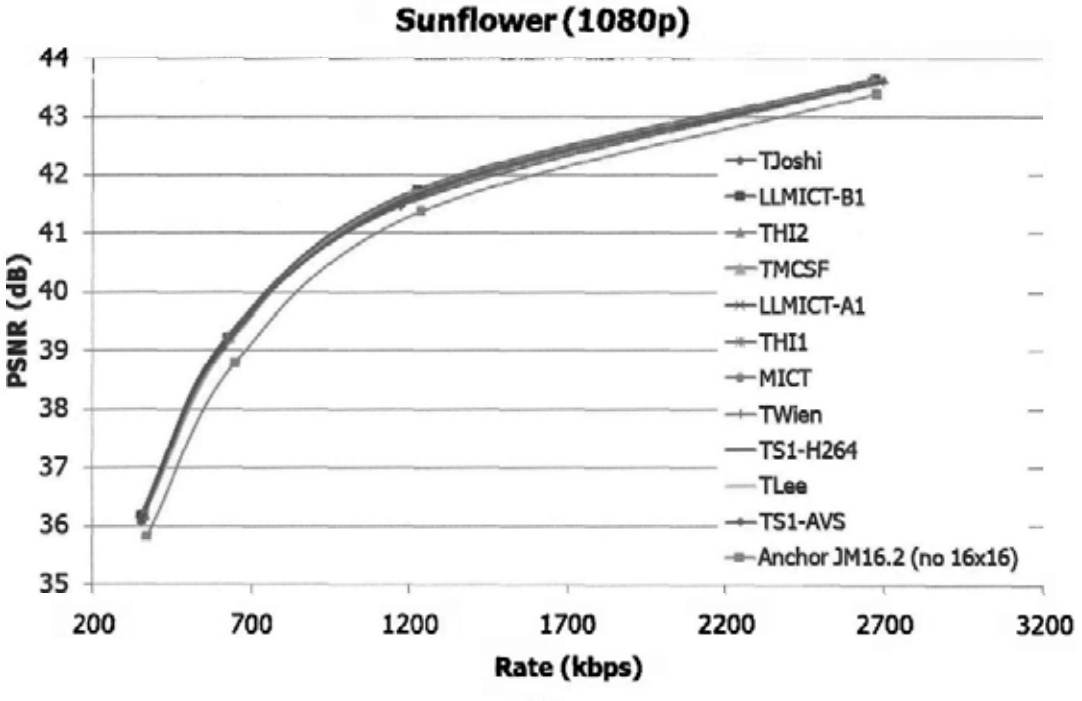


(b)

Figure 3-11 RD curves for (a) BlowingBubble (WQVGA) and (b) BaskballPass (WVGA)



(a)



(b)

Figure 3-12 RD curves for (a) Crew (720p) and (b) Sunflower (1080p)

3.7.2 Subjective Evaluation

In this section, the subjective evaluation of the video sequences coded with different order-16 transforms will be given. Several sequences of different resolutions will be illustrated. Each of them is coded with different transforms individually. In order to show their differences clearly, cropped version will be shown instead of the whole frame.

Figure 3-13 shows the 84th frame of the sequence "BasketballPass" in WQVGA (416×240) coded with different transforms and QP = 32. When it is not coded with order-16 transform, the fine details of the image are destroyed. The lines on the wall (upper left) and on the door (upper right) almost disappear in Figure 3-13 (b). They are preserved when coded with the order-16 transform ((c)-(l)). Order-16 transform preserves more detail such as the player's face (bottom right). It is also observable that the use of order-16 transform reduces the ringing artifacts near the high contrast regions (bottom left).

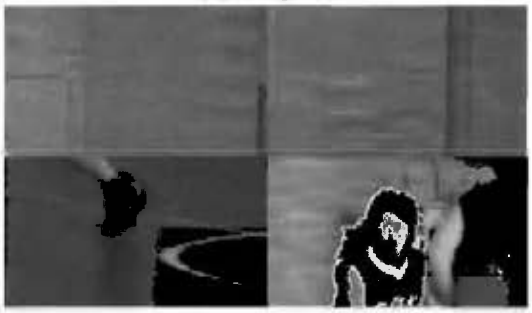
The 270th frame of "BQMall" in WVGA coded at QP = 32 is shown in Figure 3-14. The lines on the wall disappear in the decoded images (upper left). Only those coded with (j) $\mathbf{T}_{Joshi}$, (k) LLMICT-A1 and (l) LLMICT-B1 can preserve these details. The characters (upper right) and the old man's face (bottom left) are blurred in different degree. It is shown that it suffers most when it is coded without order-16 transform. The texture of the lady's hat (bottom right) can hardly be seen without order-16 transform. It is preserved with order-16 transform, especially with (j) $\mathbf{T}_{Joshi}$ and (k) LLMICT-A1.

Figure 3-15 shows the 100th frame of "SpinCalendar" in 720p coded at QP = 27. The differences among different transforms in this sequence are not obvious. The pictures on the cans (top left) are slightly sharper when order-16 transform is used. Less ripple artifacts are observed near the letters on the calendar (top right). It is quite clear that more texture is preserved on the field (bottom).

These examples show that order-16 transforms help to preserve the details of the pictures and to reduce the artifacts. However, the differences among the picture coded with different order-16 transforms are not obvious. Their subjective performances are more or less the same. In general, if the transform has a better objective performance, such as LLMICT-A1, its subjective performance is slightly better.

(a) Original

(b) Anchor



(c) $T_{S1\text{-}AVS}$

(d) $T_{S1\text{-}H264}$

(e) MICT

(f) T_{HI1}

(g) T_{HI2}

(h) T_{Wien}

(i) T_{Lee}

(j) T_{Joshi}

(k) LLMICT-A1

(l) LLMICT-B1

(m) MCSFICT

Figure 3-13 Subjective quality of "BasketballPass (WQVGA)", 84th frame, coded at QP = 32.

(a) Original

(b) Anchor

(c) $\mathbf{T}_{SI\text{-}AVS}$

(d) $\mathbf{T}_{SI\text{-}H264}$

(e) MICT

(f) $\mathbf{T}_{HI1}$

(g) $\mathbf{T}_{HI2}$

(h) $\mathbf{T}_{Wien}$

(i) $\mathbf{T}_{Lee}$

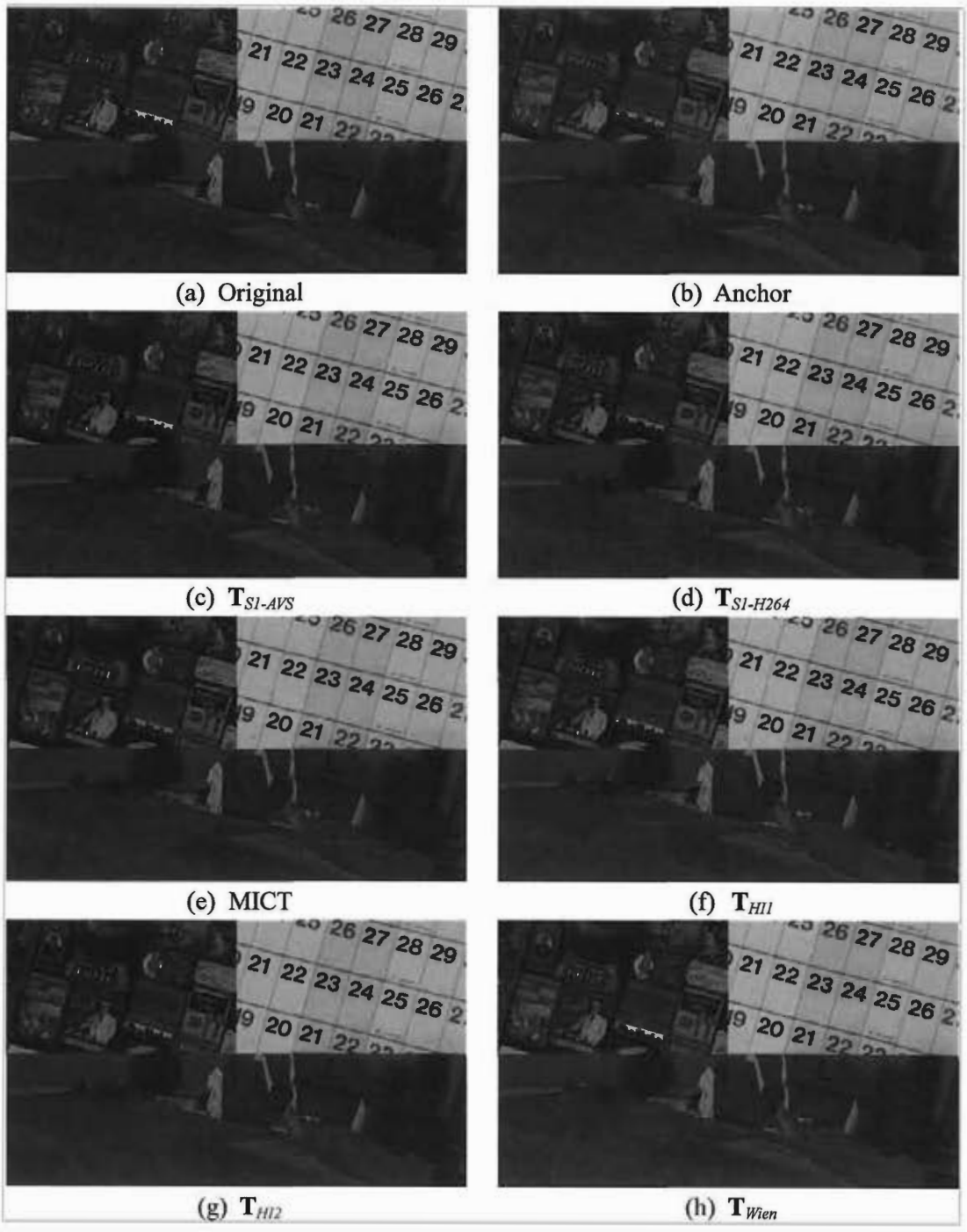
(j) $\mathbf{T}_{Joshi}$

(k) LLMICT-A1

(l) LLMICT-B1

(m) MCSFICT

Figure 3-14 Subjective quality of "BQMall (WVGA)", 270th frame, coded at QP = 32.



(a) Original (b) Anchor

(c) $\mathbf{T}_{S1\text{-}AVS}$ (d) $\mathbf{T}_{S1\text{-}H264}$

(e) MICT (f) $\mathbf{T}_{HI1}$

(g) $\mathbf{T}_{HI2}$ (h) $\mathbf{T}_{Wien}$

(i) $\mathbf{T}_{Lee}$

(j) $\mathbf{T}_{Joshi}$

(k) LLMICT-A1

(l) LLMICT-B1

(m) MCSFICT

Figure 3-15 Subjective quality of "SpinCalendar (720p)", 100th frame, coded at QP = 27.

3.7.3 Usage of Order-16 transform

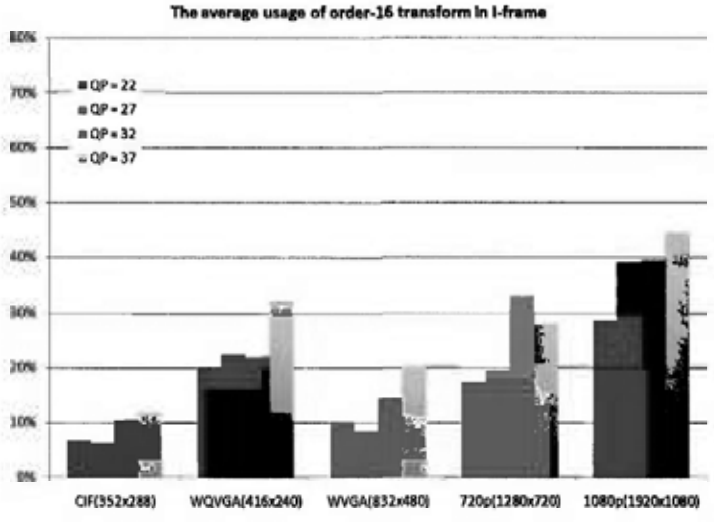
In our implementation, the transform size selection is according to the RD-performance. The one with the best RD-performance is chosen. If order-16 transform is more frequently chosen, a higher gain is provided. This also implies its importance. The usage analysis of order-16 transform focuses on our proposed LLMICT-A1 only. Its statistics is shown in Figure 3-16. The average usages of order-16 transform in different frame resolution at different QP are shown. The analysis here is in 3 aspects:

- **Frame Type:** it is observed that P-frame has the highest usage percentage (38~78%). The usage in I-frame is from 6% to 45%. It is from 15% to 35% in B-frame. They all show a pretty high usage of order-16 transform in different frame types. The usage of order-16 transform in I-frame depends on the frame nature. If it has a larger portion of homogenous region, the usage will be higher. For example, "Raven" in 720p has a large portion of smooth background. Its order-16 transform usage in I-frame is around 40%. In contrast, the usage is only 20% in highly textured sequence "City" in 720p. The usage in B-frame is relatively low because the hierarchical-B prediction structure in H.264/AVC offers a very good prediction. The predicted residue is so small that many MB are coded in skip mode (~75%). In those "non-skip" MB, around 90% MB are coded with order-16 transform. This is a very high percentage.

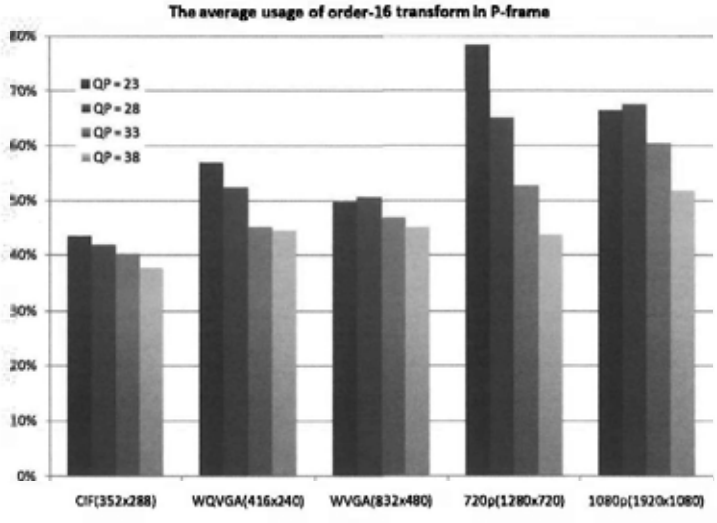
- **QP:** The usage changes differently in different frame types as QP increases. The usage increases as the QP increases in I-frame. This is because as QP increases, the RDO targets a lower rate and tolerate larger distortion. Larger transform has a

higher compressibility and hence it is favored. An obvious decreasing trend is observed in P-frame as the QP rises. This is because the larger QP makes more MB become all-zero blocks such that skip mode is favored. This lowers the usage of order-16 transform. A very slow decreasing trend is observed in B-frame. The usage of order-16 transform is relatively insensitive to the QP change in B-frame. This is because the usage of order-16 transform in B-frame is lower than in P-frame. The usage reduction of order-16 transform in B-frame is not as signification as in P-frame.

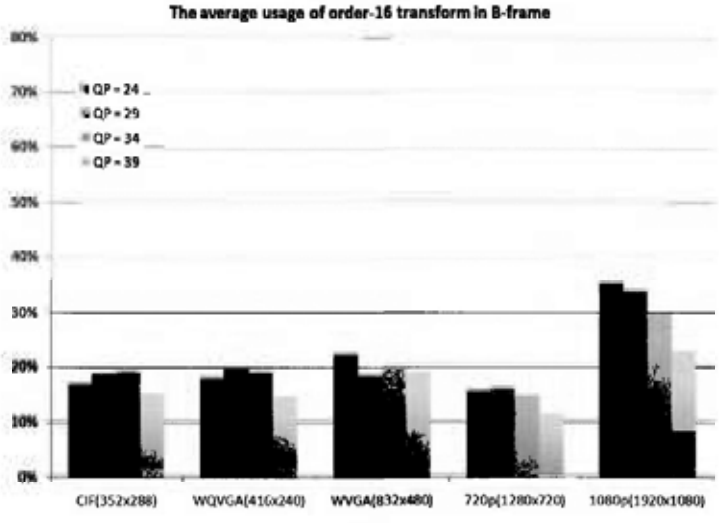
■ **Resolution:** It can be observed that the usage of order-16 transform is in an increasing trend as the resolution increases. It is more obvious in I-frame than in P- and B-frames. It can be observed that the average usage in I-frame is over 30% in 1080p while it is not more than 10% in CIF. This is because as the resolution increases, the chance of a MB covering a smooth area increases. Larger order transform has a better coding performance for smoother blocks and hence more MB prefers to be coded with order-16 transform. In P- and B-frames, the increasing trend is not that obvious. However, we can still observe that the average usage increase from 40% to over 50% in P-frames and from around almost 20% to 30% in B-frames. These shows the importance of order-16 transform in HD sequences. It is expected that there will be higher usage and more gain provided in sequences with even higher resolutions.



(a)



(b)



(c)

Figure 3-16 The average usage of order-16 in (a) I-frame, (b) P-frame and (c) B-frame.

3.7.4 Gain from Order-16 transform

In Section 3.7.1 the BD bit rate and the BD PSNR mentioned in [14] are computed. These are the overall gains in the ABT system. However, not all MB are coded with order-16 transform and the gain from order-16 transform is diluted by those coded by order-8 and order-4 transform. We are also interested in how much gain solely comes from order-16 transform. That is if a MB is coded with order-16 transform rather than order-8 or other-4 transform, how much gain should be obtained. Assume the sequence header and the frame header information is negligible comparing with the MB information. The bit rate for a coded sequence, R, can be divided into two parts: coded by order-16 transform and non-order-16 transform, denoted as R_{o16} and R_{n16} respectively. Suppose the PSNR is the same, the delta bit rate can be divided into two parts:

$$\Delta bitrate = \frac{R_{o16}}{R_{o16} + R_{n16}} \Delta bitrate_{o16} + \frac{R_{n16}}{R_{o16} + R_{n16}} \Delta bitrate_{n16}. \tag{3.19}$$

We may assume there is very little or even no gain from non-order-16 transform, i.e. $\Delta bitrate_{n16} \approx 0$. Therefore $\Delta bitrate_{o16}$ becomes:

$$\Delta bitrate_{o16} \approx \frac{\Delta bitrate}{R_{o16} / (R_{o16} + R_{n16})}. \tag{3.20}$$

Using the idea of calculating BD bit rate (see Chapter 1 for more detail), the BD bit rate purely comes from order-16 transform, $BD\text{-}bitrate_{o16}$ is:

$$BD\text{-}bitrate_{o16} = \int_{P_{lower}}^{P_{upper}} \Delta bitrate_{o16}\, dp \approx \frac{BD\text{-}bitrate}{w_{o16}(p)} \tag{3.21}$$

where

$$w_{o16}(p) = \frac{1}{P_{upper} - P_{lower}} \int_{P_{lower}}^{P_{upper}} \frac{R_{o16}(p)}{R_{o16}(p) + R_{n16}(p)} dp\,. \tag{3.22}$$

which is a function of PSNR. Similarly, the BD PSNR purely comes from order-16 transform, $BD\text{-}bitrate_{o16}$ is:

$$BD\text{-}PSNR_{o16} = \int_{R_{lower}}^{R_{upper}} \Delta PSNR_{o16}\, dr \approx \frac{BD\text{-}PSNR}{w_{o16}(r)} \tag{3.23}$$

where

$$w_{o16}(r) = \frac{1}{R_{upper} - R_{lower}} \int_{R_{lower}}^{R_{upper}} \frac{R_{o16}(r)}{R_{o16}(r) + R_{n16}(r)} dr\,. \tag{3.24}$$

The gains solely from order-16 transform are listed in Table 3-7 and Table 3-8. $w_{o16}(r)$ and $w_{o16}(p)$ are the average percentage of the bit rate allocated to order-16 transform over the tested bit rate range and the tested PSNR range respectively. From these values, it can be observed that a high percentage of bits are allocated to order-16 transform (from 33% to 79%). It can also be noticed that as the resolution increases, these values increases. This shows that order-16 transform dominates as the resolution increases. In Table 3-8, the average value of $BD\text{-}PSNR_{o16}$ is around 0.497 dB. It is very close to difference between the transform coding gains of order-16 DCT and order-8 DCT at $\rho = 0.9$.

Sequences		*BD-bitrate* (%)	$w_{o16}(p)$	$BD\text{-}bitrate_{o16}$ (%)
CIF	Foreman	-4.78	0.548	-8.72
	Mobile	-5.98	0.413	-14.48
	News	-6.22	0.377	-16.50
	Paris	-6.29	0.244	-25.78
	Tempete	-4.30	0.447	-9.62
WQVGA	BasketballPass	-5.59	0.336	-16.64
	BlowingBubble	-7.90	0.340	-23.24
	BQSquare	-5.27	0.392	-13.44
	Flowervase	-6.40	0.670	-9.55
WVGA	BasketballPass	-9.91	0.490	-20.22
	BQMall	-7.99	0.494	-16.17
	PartyScene	-9.25	0.436	-21.22
	RaceHorses	-3.08	0.446	-6.91
720p	Bigship	-3.80	0.654	-5.81
	City	-4.19	0.687	-6.10
	Crew	-7.65	0.608	-12.58
	Night	-3.92	0.490	-8.00
	Raven	-5.40	0.726	-7.44
	Parkrun	-5.22	0.576	-9.06
	ShuttleStart	-4.30	0.643	-6.69
1080p	Cactus	-6.00	0.585	-10.26
	Sunflower	-11.28	0.736	-15.33
	BasketballDrive	-7.72	0.584	-13.22
	Kimono1	-8.15	0.793	-10.28
	Pedestrian	-8.77	0.681	-12.88
	Station2	-4.98	0.780	-6.38
	RushHour	-3.85	0.775	-4.97
Average		-6.23	0.554	-12.28

Table 3-7 Delta bit rate solely from order-16 transform ($BD\text{-}bitrate_{o16}$, %)

Sequences		*BD-PSNR* (dB)	$w_{o16}(r)$	$BD\text{-}PSNR_{o16}$ (dB)
CIF	Foreman	0.21	0.542	0.39
	Mobile	0.26	0.416	0.63
	News	0.33	0.376	0.88
	Paris	0.33	0.250	1.32
	Tempete	0.17	0.456	0.37
WQVGA	BasketballPass	0.27	0.346	0.78
	BlowingBubble	0.31	0.347	0.89
	BQSquare	0.19	0.394	0.48
	Flowervase	0.32	0.667	0.48
WVGA	BasketballPass	0.41	0.493	0.83
	BQMall	0.34	0.487	0.70
	PartyScene	0.41	0.442	0.93
	RaceHorses	0.12	0.444	0.27
720p	Bigship	0.10	0.654	0.15
	City	0.12	0.697	0.17
	Crew	0.19	0.600	0.32
	Night	0.13	0.484	0.27
	Raven	0.20	0.719	0.28
	Parkrun	0.20	0.589	0.34
	ShuttleStart	0.13	0.638	0.20
1080p	Cactus	0.17	0.562	0.30
	Sunflower	0.43	0.720	0.60
	BasketballDrive	0.23	0.567	0.41
	Kimono1	0.33	0.790	0.42
	Pedestrian	0.35	0.666	0.53
	Station2	0.25	0.760	0.33
	RushHour	0.12	0.765	0.16
Average		0.245	0.551	0.497

Table 3-8 Delta PSNR solely from order-16 transform ($BD\text{-}PSNR_{o16}$, dB)

3.8 Conclusions

In this chapter, H.264/AVC is reviewed. The order-16 transforms proposed in the last chapter are integrated into its reference software JM16.2. Syntax structure is changed and the implementation details are described. New syntax elements and context models for CABAC are added. The proposed platform is tested. Different order-16 integer transforms are tested and compared. It is found that more details and less distortion are observed when order-16 transform is used. It is noticed that order-16 transform helps to reduce the artifacts. Subjective qualities between different order-16 transforms are similar. In the proposed platform, $\mathbf{T}_{joshi}$ gives the largest average bit rate reduction (6.21%). A maximum reduction of 12.11% is obtained. The proposed LLMICT-A1, MCSFICT and LLMICT-B1 are not far from it. Their average bit rate reductions are 6.18%, 6.13% and 5.97% respectively but they require fewer computations.

It is found that the usage of order-16 transform is pretty high. It is up to 78% in P-frame on average. It is also noticed that the usage of order-16 is in an increasing trend as the picture resolution increases. That is why order-16 transforms bring larger gains in HD sequences.

3.9 References

[1] T. Wiegand, G. J. Sullivan, G. Bjøntegaard and A. Luthra, "*Overview of the H.264/AVC video coding standard,*" IEEE Trans on CASVT, vol. 13, no. 7, pp 560-576, 2003.

[2] Soon-kak Kwon, A. Tamhankar and K. R. Rao, "*Overview of H.264/MPEG-4 part 10,*" Journal of Visual Communication and Image Representation, vol. 17, no. 2, pp 186-216, April 2006.

[3] D. Marpe, T. Wiegand and S. Gordon, "*H.264/MPEG4-AVC fidelity range extensions: tools, profiles, performance and application areas,*" IEEE ICIP 2005, vol. 1, pp I - 593-596, 2005.

[4] G. J. Sullivan, P. Topiwala and A. Luthra, "*The H.264/AVC Advanced Video Coding Standard: Overview and Introduction to the Fidelity Range Extensions,*" Proc. SPIE Conference on Applications of Digital Image Processing XXVII, Nov. 2004.

[5] H. Schwarz, D. Marpe and T. Wiegand, "*Overview of the Scalable Video Coding Extension of the H.264/AVC Standard,*" IEEE Trans. on CASVT, vol. 17, no. 9, pp 1103-1120, 2007.

[6] M. Wien, H. Schwarz and T. Oelbaum, "*Performance Analysis of SVC,*" IEEE Trans. on CASVT, vol. 17, no. 9, pp 1194-1203, 2007.

[7] H. Schwarz and M. Wien, "*The Scalable Video Coding Extension of the H.264/AVC Standard,*" IEEE Signal Processing Magazine, vol. 25, no. 2, pp 135-141, 2008.

[8] T. C. Thang, J. Kim, J. W. Kang and J. Yoo, "*SVC adaptiation: Standard tools and supporting methods,*" Signal Processing: Image Communication, vol. 24, no. 3, pp 214-228, 2009.

[9] Yo-Sung Ho and Kwan-Jung Oh, "*Overview of Multi-view Video Coding,*" the 14th International Workshop on Systems, Signals and Image Processing and the 6th EURASIP Conference focused on Speech and Image Processing, Multimedia Communications and Services, pp 5-12, 2007.

[10] G. J. Sullivan, "*Standards-based approaches to 3D and multiview video coding,*" Proc. SPIE, vol. 7443, Applications of Digital Image Processing XXXII, Sept. 2009.

[11] P. Merkle, K. Müller and T. Wiegand, "*3D video coding: an overview of present and*

upcoming standards," Proc. SPIE, vol. 7744, Visual Communications and Image Processing, July 2010.

[12] H.264/AVC Reference Software JM16.2. [Online] Available: http://iphome.hhi.de/suehring/tml/download/old_jm/jm16.2.zip

[13] T. K. Tan, G. Sullivan and T. Wedi, "*Recommended Simulation Common Conditions for Coding Efficiency Experiments Revision 4,*" Document VCEG-AJ10, Oct. 2008. [Online] Available: http://wftp3.itu.int/av-arch/video-site/0810_San/

[14] G. Bjøntegaard, "*Calculation of Average PSNR Differences between RD-curves,*" ITU-T SG16/Q6, Document VCEG-M33, April 2001. [Online] Available: http://wftp3.itu.int/av-arch/video-site/0104_Aus/

[15] R. Joshi, Y. Reznik, and M. Karczewicz, "*Simplified Transforms for Extended Block Sizes,*" ITU-T SG16/Q6, document VCEG-AL30, July 2009. [Online] Available: http://wftp3.itu.int/av-arch/video-site/0906_LG/

[16] Y. Ye and M. Karczewicz, "*Improved Intra Coding,*" ITU-T SG16/Q6, document VCEG-AG11, Oct. 2007. [Online] Available: http://wftp3.itu.int/av-arch/video-site/0710_She/

Chapter 4 ABT in AVS

4.1 Overview of AVS

Audio Video Standard (AVS) [1]-[4] is a digital audio and video compression standard developed by the workgroup of the same name in mainland China. The standard development started in 2002. The standard is divided into several parts and each part specifies an application area. AVS Part 2 and Part 7 relate to video coding in different aspect. Part 2 targets in HD digital video broadcasting and storage while Part 7 targets in lower complexity, lower resolution mobile applications. Similar to H.264/AVC, the standard defines different profiles specifying subset of the coding tools. There are 4 main profiles and their target applications are:

- Jizhun (base, 基準) profile: video broadcasting,
- Jiben (basic, 基本, also called Yidong, 移動) profile: mobile applications,
- Shenzhan (extended, 伸展) profile: video surveillance, and
- Jiaqiang (enhanced, 加強) profile: multimedia entertainment.

The AVS-Part 2 Jizhun profile was approved as national standard in 2006. Recently, more profiles are added to the standard to enhance the functionality. For example, Shuangmu Liti Jizhun (Stereo 3D base, 雙目立體基準) profile was added to support stereo 3D video coding. In order to improve the coding performance of the existing standard, the drafting of the next generation standard, AVS 2.0, started in March 2009. It was expected to finish the first working draft in July 2010. It is still based on the hybrid coding architecture. It aims to provide significant improvement on the top of AVS Part 2. It will support super high resolution up to 8k×4k and higher color bit depth up to 14 bits. The final committee draft is targeted to be finished in 2012 [5][6].

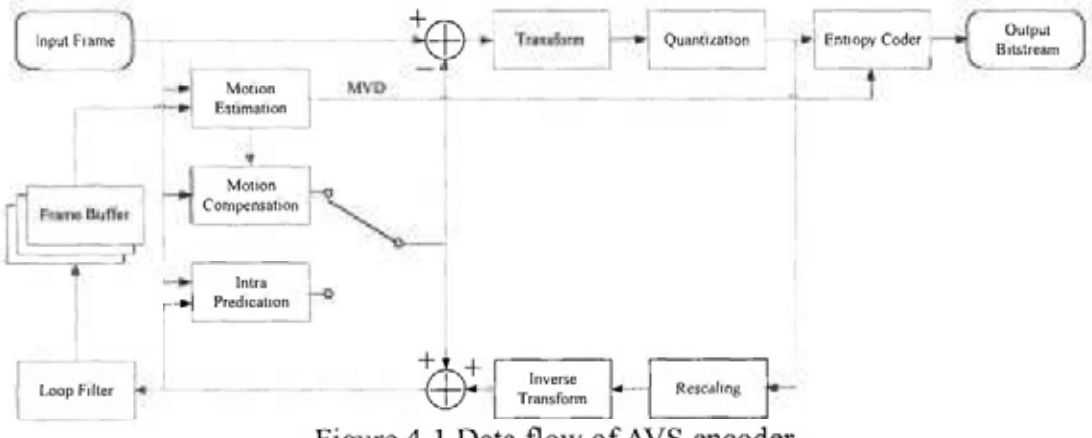


Figure 4-1 Data flow of AVS encoder.

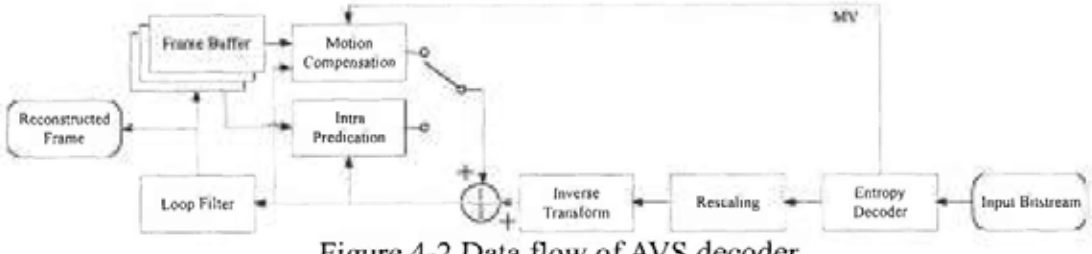


Figure 4-2 Data flow of AVS decoder.

In this thesis, our focus is on AVS Part 2 which targets in HD video coding. It is expected that our techniques are portable to AVS 2.0 and acts as one of the coding

tools in the new standard.

Same as H.264/AVC, AVS encoder is also a hybrid video coder. The structure of the encoder and the decoder are the same as H.264/AVC as shown in Figure 4-1 and Figure 4-2. However, the details in many functional blocks are different from H.264/AVC. The following are some of the typical techniques adopted in AVS Part 2:

■ **Intra prediction:** 4×4 and 8×8 intra predictions are available in AVS standard. However, 4×4 intra prediction is only adopted in Jiben Profile while 8×8 intra prediction is adopted in the other 3 main profiles. There are 9 different intra prediction modes for 4×4 intra prediction. They are similar to those specified in H.264/AVC. In contrast, there are only 5 prediction modes for 8×8 intra prediction. The five modes are Vertical, Horizontal, DC, Down-Left and Down-Right. This lowers the complexity of the encoder but maintains high coding efficiency.

■ **Inter prediction:** There are only 4 different partition sizes for inter prediction. They are 16×16, 16×8, 8×16 and 8×8. Also, experiments show that the nearest 2 decoded frames are the most referenced frames. The frequency of referencing further frames is significantly lower than these 2 frames. As a result, the maximum number of reference frames in AVS is 2. This reduces not only the motion estimation complexity, but also the side information indicating the reference frames.

■ **Simplified interpolation filter:** Quarter-pixel motion estimation is allowed in AVS. In H.264/AVC, 6-tap interpolation filter is used. In AVS, two 4-tap interpolation filters cascading together are used. One is for half pixel accuracy

while the other is for quarter-pixel. The computation complexity is lowered.

■ **Pre-scaled ICT:** Low complexity 8×8 ICT is adopted in Jizhun, Shenzhan and Jiaqiang profiles while 4×4 ICT is adopted in Jiben profile. They are organized in pre-scaled [7] structure such that the scaling matrices are only located in encoder side. In H.264/AVC, the scaling matrices are located equally in both encoder and decoder.

■ **CA-2D-VLC:** The quantized coefficients are specially coded with a Context-based Adaptive 2D Variable Length Coder (CA-2D-VLC) [8]. It is a low complexity entropy coder which utilizes the joint probability of level-run combination. As a result, a high coding efficiency is obtained.

■ **Enhanced Arithmetic Coding (EAC):** It is an arithmetic coder with logarithmic probability model. Its probability estimation is multiplication-free such that it has lower complexity than the CABAC in H.264 but higher coding efficiency than CA-2D-VLC. Its coding efficiency is comparable with CABAC.

Based on the AVS architecture, a joint proposal [9] was submitted and accepted into the AVS reference software. In this proposal, researchers from Tsing Hau University proposed the use of 4×4 ICT adopted in Jiben profile. On the other hand, we proposed order-16 Simple Integer Transform ($\mathbf{T}_{SI\text{-}AVS}$). Together with the existing 8×8 ICT, an Arbitrary Block-size Transform (ABT) system, integrated with 4×4, 8×8 and 16×16 transforms, is formed. A maximum bit reduction of 19.35% is reported. On the top of this system, we have proposed several methods to improve the coding efficiency [10]-[13]. The proposal [13] was adopted into the latest AVS

reference software.

In this thesis, the reference software RM6.2*l* acts as the testing platform. The order-16 transforms mentioned in Chapter 2 will be integrated into it. Details of this platform will be described from Section 4.2 to 4.7. Experiment and analysis will be shown in Section 4.8. Finally, conclusions will be drawn in 4.9

4.2 Intra prediction

In AVS, only 4×4 and 8×8 intra predictions are present. There is no 16×16 intra prediction in the standard. The 16×16 intra-prediction proposed in [14] is integrated in our platform. It has 5 different prediction modes (Figure 4-3). They are the vertical, horizontal, DC, down-left and down-right prediction. The referenced pixels from previous decoded blocks are low-pass filtered before prediction. When 16×16 intra-prediction is used, 16×16 transform is used compulsorily.

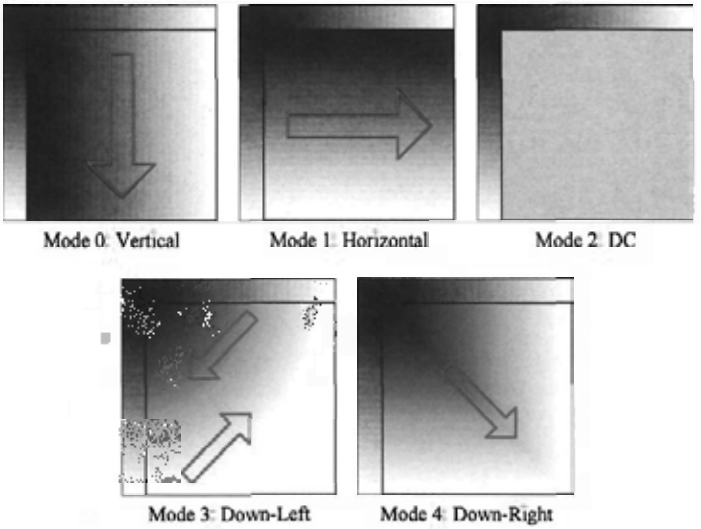


Figure 4-3 16×16 intra prediction in proposed AVS platform.

4.3 Transforms

4.3.1 ABT in AVS

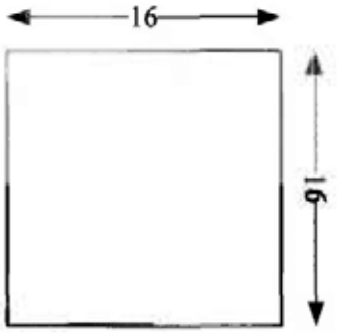
Order-8 ICT was adopted in AVS Part 2. Its integer kernel is shown in (4-1). Order-4 ICT was also adopted in AVS Part 7. Its integer kernel is shown in (4-2). Researchers from Tsing Hau University proposed to form an ABT system with these two transforms in [15]. At the same time, we proposed an ABT platform using order-16 and order-8 transforms [9]. In the proposal, order-16 Simple Integer Transform $\mathbf{T}_{SI\text{-}AVS}$ was proposed. These two ideas were merged together and an ABT platform including 4×4, 8×8 and 16×16 transforms is formed [16]. This proposal was accepted by the AVS workgroup and integrated into the reference software.

$$E_{IC}^{(8)} = \begin{bmatrix} 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 \\ 10 & 9 & 6 & 2 & -2 & -6 & -9 & -10 \\ 10 & 4 & -4 & -10 & -10 & -4 & 4 & 10 \\ 9 & -2 & -10 & -6 & 6 & 10 & 2 & -9 \\ 8 & -8 & -8 & 8 & 8 & -8 & -8 & 8 \\ 6 & -10 & 2 & 9 & -9 & -2 & 10 & -6 \\ 4 & -10 & 10 & -4 & -4 & 10 & -10 & 4 \\ 2 & -6 & 9 & -10 & 10 & -9 & 6 & -2 \end{bmatrix}, \tag{4-1}$$

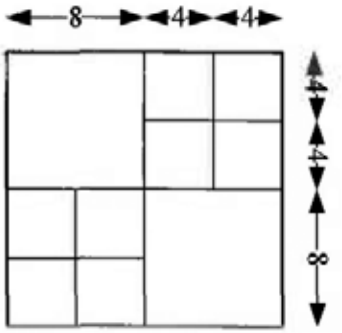
$$E_{IC}^{(4)} = \begin{bmatrix} 2 & 2 & 2 & 2 \\ 3 & 1 & -1 & -3 \\ 2 & -2 & -2 & 2 \\ 1 & -3 & 3 & -1 \end{bmatrix} \tag{4-2}$$

In [16], researchers from Tsing Hau University also proposed a combinative coding manner. When a MB is not coded with 16×16 transform, it can be coded with 8×8 transform together with 4×4 (Figure 4-4) at the same time. There are 16 different

combinations. The selection between 4×4 and 8×8 transform is in sub-block basis. It is decided when every 8×8 sub-block is coded. The best combination is selected by RD optimization. When a MB is coded with order-16 transform, this means that if offers a RD cost lower than any combination of 8×8 and 4×4 transform. An example is shown in Figure 4-5. We can see that the transform size for coding smooth regions changes from 8×8 to 16×16. We can also notice that the rate reduces and the PSNR increases when 16×16 transform is enabled.



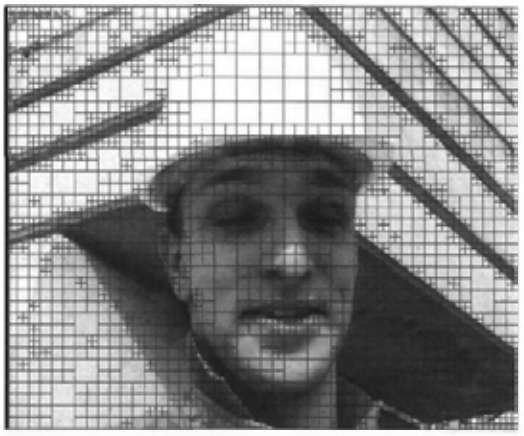
A MB coded with 16×16 Transform



A MB coded with Combinative Transform, Mixture of 4×4 and 8×8 Transform

Figure 4-4 Combinative ABT in proposed AVS platform.

(a) 38208 bits, 35.589dB



(b) 38064 bits, 35.643dB

Figure 4-5 ABT (a) without 16×16 Transform, and (b) with 16×16 Transform.

In this chapter, some order-16 transforms described in Chapter 2 will be integrated to the AVS ABT platform. They are tested individually. These transforms include:

- Simple Integer Transform, $\mathbf{T}_{SI\text{-}AVS}$,
- MICT,
- Hybrid Integer Transform, $\mathbf{T}_{HI1}$,
- Hybrid Integer Transform, $\mathbf{T}_{HI2}$,
- The integer transform proposed by Lee, $\mathbf{T}_{Lee}$,
- The integer transform proposed by Joshi, $\mathbf{T}_{Joshi}$,
- LLMICT-A1, $\mathbf{T}_{LLM\text{-}A1}$,
- LLMICT-B1, $\mathbf{T}_{LLM\text{-}B1}$, and
- Modified CSFICT (MCSFICT), $\mathbf{T}_{MCSF}$.

4.3.2 Flexible Transform Size Selection

In RM 6.2*l* and before, the transform size selection is limited. In I-frame, only order-4 and order-8 transform were allowed. In B-frame, only order-8 and order-16 transform were allowed. These three transforms only coexisted in P-frames. It was because the earlier research showed low usage of order-16 transform in I-frame and low usage of order-4 transform in B-frame. However, we found that this is not true, especially in HD sequences. The usage of order-16 transform in I-frame is quite high (See Section 4.8.3). As a result, we proposed a Flexible Transform Size Selection. These three transforms can be selected flexibly based on their RD cost. The optimal transform can be used without any restriction. We found that this modification brings an average bit rate reduction of 4% in 1080p sequences [13].

4.3.3 Transform Design Constraints in AVS

It is more challenging in designing integer transform for AVS. There are more constraints in its design and the integer kernel must be suitably adjusted before implementing to the AVS reference software. The transform process adopted in AVS is the pre-scaled integer transform (PIT) [7]. The forward transform and the inverse transform processes are modeled in (4-3) and (4-4) respectively. Suppose the integer kernel is **E**. **X** and **Y** are the input pixels and the reconstructed pixels respectively. Recall that the 2-D scaling matrix **S**, which is a combination of 1-D scaling matrix **K**. **S** locates in both forward and inverse transform process in H.264/AVC. When **S** only locates in the forward transform process, it is called pre-scaled transform. In contrast, if the scaling matrix **S** only locates in the inverse transform process, it is called post-scaled transform and it is adopted in VC-1 [17]. No matter pre-scaled or post-scale transform, the transform coefficients are scaled.

$$\begin{aligned}\mathbf{F}_{PIT} &= \left(\mathbf{E}\,\mathbf{X}\,\mathbf{E}^{T}\right)\odot\mathbf{S}\odot\mathbf{S} \\ &= \mathbf{G}\odot\mathbf{S}\odot\mathbf{S}\end{aligned} \quad \text{(4-3)}$$

$$\mathbf{Y} = \mathbf{E}^{T}\,\mathbf{F}_{PIT}\,\mathbf{E}. \quad \text{(4-4)}$$

There is another constraint in AVS. The transform, quantization and rescaling process must be able to be implemented on a 16-bit system. Therefore, the precisions of the transform coefficients are limited. To prevent overflow, the transform coefficients are rounded. Those with smaller dynamic ranges may suffer bigger rounding error in fixed point implementation. As a result, its coding performance may be lowered. Suggestions were provided in [7] for the design of pre-scaled transform so as to prevent such degradation. Weighting Factor

Difference (WFD) was defined in [7]:

$$WFD = \frac{\max_{i,j}(S(i,j))}{\min_{i,j}(S(i,j))}. \tag{4-5}$$

It measures the extreme dynamic range ratio of the coefficients in a given transform. The larger WFD, there are bigger difference between dynamic ranges of different transform coefficients. WFD is minimum (= 1) when all $S(i, j)$ are the same. This means all basis vectors have the same norm. In order to keep a good performance, the transform should be unitary and it is suggested that the upper bound for WFD to be $\sqrt{2}$ in [7]. To fulfill this requirement, the basis vectors in the transform are adjusted with integer factors, m_i. The integer kernel becomes:

$$\mathbf{E} = \begin{bmatrix} [e_{00} & e_{01} & \cdots & e_{0(N-1)}]m_0 \\ [e_{10} & e_{11} & \cdots & e_{1(N-1)}]m_1 \\ & & \vdots & \\ [e_{(N-1)0} & e_{(N-1)1} & \cdots & e_{(N-1)(N-1)}]m_{(N-1)} \end{bmatrix}. \tag{4-6}$$

After the adjustment, the waveforms remain unchanged. However, the choice of these factors highly affects the coding performance. It is not only the WFD issue, but also the usage of the dynamic range. An inefficient use of the coefficient dynamic range lowers the performance. The DC coefficient, $F_{PIJ}(0, 0)$, which has the largest dynamic range, should have a maximum magnitude of power of 2 so as to maximize the dynamic range usage. This can be achieved by setting the norm of the DC basis vector to be 2^n. As a result, the minimum integer scaling factors, m_i, are selected such that the following two criteria are fulfilled in our implementation:

- WFD must be less than a threshold WFD_{max}.

- The magnitude of the norm of the DC basis vector should be of 2^n so as to maximize the usage of dynamic range.

By so doing, the dynamic ranges of the coefficients are restricted in a narrow range as shown in Figure 4-6. In our case, WFD_{max} is 1.6.

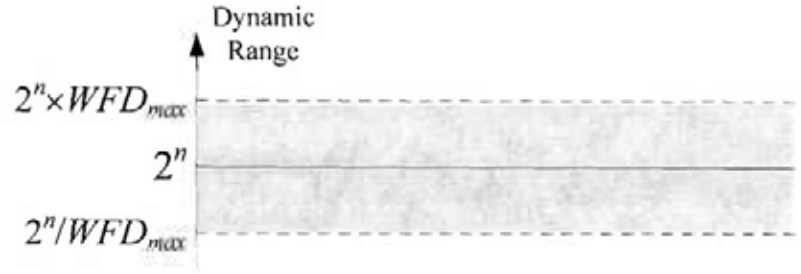


Figure 4-6 the dynamic range of the coefficient after adjustment.

As a result, the integer kernel of $\mathbf{T}_{Lee}$, $\mathbf{T}_{Joshi}$, LLMICT-A1 and LLMICT-B1 are shown in (4-7) to (4-10) respectively. The WFD after adjustment are shown in Table 4-1. Experiment shows that this adjustment is very important in PIT design. A bit rate reduction of more than 7% is observed before and after the adjustment with the same transform.

Transform	Weighting Factor Difference (WFD)	
	Before Adjustment	After Adjustment
$\mathbf{T}_{S1\text{-}AVS}$	1.158	1.158
MICT	1.269	1.269
$\mathbf{T}_{HI1}$	1.245	1.245
$\mathbf{T}_{HI2}$	1.246	1.246
$\mathbf{T}_{Lee}$	2.390	1.578
$\mathbf{T}_{Joshi}$	2.716	1.149
LLMICT-A1	2.173	1.241
LLMICT-B1	2.000	1.471
MCSFICT	1.179	1.179

Table 4-1 The Weighting Factor Difference of different transform.

$$E_{16\times 16}=\begin{bmatrix}
64 & 64 & 64 & 64 & 64 & \cdots & 64 & 64 & 64 & 64\\
108 & 100 & 112 & 84 & 84 & \cdots & -84 & -112 & -100 & -108\\
100 & 84 & 56 & 20 & -20 & \cdots & 20 & 56 & 84 & 100\\
100 & 84 & 12 & -36 & -112 & \cdots & 36 & -12 & -84 & -100\\
102 & 42 & -42 & -102 & -102 & \cdots & -102 & -42 & 42 & 102\\
112 & 12 & -84 & -100 & -44 & \cdots & 100 & 84 & -12 & -112\\
84 & -20 & -100 & -56 & 56 & \cdots & -56 & -100 & -20 & 84\\
84 & -36 & -100 & 12 & 108 & \cdots & -12 & 100 & 36 & -84\\
64 & -64 & -64 & 64 & 64 & \cdots & 64 & -64 & -64 & 64\\
84 & -112 & -44 & 108 & -12 & \cdots & -108 & 44 & 112 & -84\\
56 & -100 & 20 & 84 & -84 & \cdots & 84 & 20 & -100 & 56\\
36 & -108 & 84 & 44 & -100 & \cdots & -44 & -84 & 108 & -36\\
42 & -102 & 102 & -42 & -42 & \cdots & -42 & 102 & -102 & 42\\
44 & -84 & 108 & -112 & 36 & \cdots & 112 & -108 & 84 & -44\\
20 & -56 & 84 & -100 & 100 & \cdots & -100 & 84 & -56 & 20\\
12 & -44 & 36 & -84 & 84 & \cdots & 84 & -36 & 44 & -12
\end{bmatrix}, \tag{4-7}$$

$$E_{8\times 8}=\begin{bmatrix}
512 & 512 & 512 & 512 & 512 & \cdots & 512 & 512 & 512 & 512\\
688 & 672 & 608 & 544 & 432 & \cdots & -544 & -608 & -672 & -688\\
702 & 598 & 390 & 30 & -130 & \cdots & 130 & 390 & 598 & 702\\
681 & 417 & 87 & -348 & -609 & \cdots & 348 & -87 & -417 & -681\\
670 & 268 & -268 & -670 & -670 & \cdots & -670 & -268 & 268 & 670\\
609 & 87 & -543 & -672 & -201 & \cdots & 672 & 543 & -87 & -609\\
592 & -148 & -703 & -407 & 407 & \cdots & -407 & -703 & -148 & 592\\
539 & -341 & -649 & 77 & 671 & \cdots & -77 & 649 & 341 & -539\\
512 & -512 & -512 & 512 & 512 & \cdots & 512 & -512 & -512 & 512\\
407 & -583 & -187 & 671 & -77 & \cdots & -671 & 187 & 583 & -407\\
407 & -703 & 148 & 592 & -592 & \cdots & 592 & 148 & -703 & 407\\
348 & -696 & 444 & 201 & -672 & \cdots & -201 & -444 & 696 & -348\\
268 & -670 & 670 & -268 & -268 & \cdots & -268 & 670 & -670 & 268\\
168 & -564 & 696 & -609 & 348 & \cdots & 609 & -696 & 564 & -168\\
130 & -390 & 598 & -702 & 702 & \cdots & -702 & 598 & -390 & 130\\
96 & -176 & 336 & -432 & 544 & \cdots & 432 & -336 & 176 & 96
\end{bmatrix}, \tag{4-8}$$

$$E_{16\times 8}=\begin{bmatrix}
256 & 256 & 256 & 256 & 256 & \cdots & 256 & 256 & 256 & 256\\
344 & 336 & 304 & 272 & -216 & \cdots & -272 & -304 & -336 & -344\\
320 & 288 & 192 & 64 & -64 & \cdots & 64 & 192 & 288 & 320\\
319 & 203 & 33 & -157 & -291 & \cdots & 157 & -33 & -203 & -319\\
320 & 128 & -128 & -320 & -320 & \cdots & -320 & -128 & 128 & 320\\
283 & 49 & -261 & -319 & -87 & \cdots & 319 & 261 & -49 & -283\\
288 & -64 & -320 & -192 & 192 & \cdots & -192 & -320 & -64 & 288\\
294 & -186 & -354 & 42 & 366 & \cdots & -42 & 354 & 186 & -294\\
256 & -256 & -256 & 256 & 256 & \cdots & 256 & -256 & -256 & 256\\
222 & -318 & -102 & 366 & -42 & \cdots & -366 & 102 & 318 & -222\\
192 & -320 & 64 & 288 & -288 & \cdots & 288 & 64 & -320 & 192\\
171 & -327 & 203 & 87 & -319 & \cdots & -87 & -203 & 327 & -171\\
128 & -320 & 320 & -128 & -128 & \cdots & -128 & 320 & -320 & 128\\
87 & -261 & 329 & -291 & 157 & \cdots & 291 & -329 & 261 & -87\\
64 & -192 & 288 & -320 & 320 & \cdots & -320 & 288 & -192 & 64\\
48 & -88 & 168 & -216 & 272 & \cdots & 216 & -168 & 88 & -48
\end{bmatrix}, \tag{4-9}$$

$$E_{16\times 4}=\begin{bmatrix}
128 & 128 & 128 & 128 & 128 & \cdots & 128 & 128 & 128 & 128\\
200 & 200 & 184 & 152 & 136 & \cdots & -152 & -184 & -200 & -200\\
160 & 144 & 96 & 32 & -32 & \cdots & 32 & 96 & 144 & 160\\
200 & 100 & 25 & -100 & -175 & \cdots & 100 & -25 & -100 & -200\\
160 & 64 & -64 & -160 & -160 & \cdots & -160 & -64 & 64 & 160\\
175 & 25 & -140 & -185 & -80 & \cdots & 185 & 140 & -25 & -175\\
144 & -32 & -160 & -96 & 96 & \cdots & -96 & -160 & -32 & 144\\
150 & -100 & -170 & 10 & 180 & \cdots & -10 & 170 & 100 & -150\\
128 & -128 & -128 & 128 & 128 & \cdots & 128 & -128 & -128 & 128\\
100 & -150 & -60 & 180 & -10 & \cdots & -180 & 60 & 150 & -100\\
96 & -160 & 32 & 144 & -144 & \cdots & 144 & 32 & -160 & 96\\
100 & -200 & 145 & 80 & -185 & \cdots & -80 & -145 & 200 & -100\\
64 & -160 & 160 & -64 & -64 & \cdots & -64 & 160 & -160 & 64\\
25 & -175 & 200 & -175 & 100 & \cdots & 175 & -200 & 175 & -25\\
32 & -96 & 144 & -160 & 160 & \cdots & -160 & 144 & -96 & 32\\
40 & -40 & 88 & -136 & 152 & \cdots & 136 & -88 & 40 & -40
\end{bmatrix}, \tag{4-10}$$

4.4 Quantization and Rescaling

After transformation, the coefficients are quantized in the encoder side and are rescaled in the decoder side. In pre-scaled transform, the two scaling matrices **S** locates in the encoder side and they are embedded into the quantization process as shown in Figure 4-7.

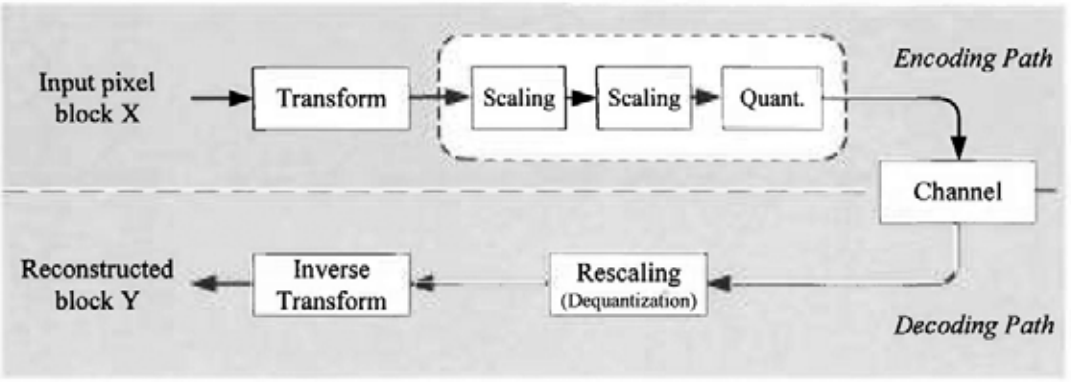


Figure 4-7 Data flow of quantization and rescaling in AVS.

4.4.1 Quantization

Recall (4-3), (4-4) and the definition of **S** in last chapter. The quantization process with step size Q_{step} and rounding offset Q_{offset} can be modeled as:

$$F_q(i,j) = sign(G(i,j)) \times \left\lfloor \frac{G(i,j) \cdot S^2(i,j)}{Q_{step}} + Q_{offset} \right\rfloor \quad (4\text{-}11)$$

Q_{offset} is defined as:

$$Q_{offset} = \begin{cases} 1/3 & \text{for intra block} \\ 1/6 & \text{for inter block} \end{cases} \quad (4\text{-}12)$$

Since division is a heavy computation load, it is replaced by multiplication with integer approximation where:

$$scale(i,j) = round\left(S^2(i,j) \times 2^{Sbit}\right) \text{ and} \tag{4-13}$$

$$Q_{tab}(QP) = round\left(\frac{2^{15}}{Q_{step}}\right) \approx round\left(2^{15-\frac{QP}{8}}\right) \text{ for } QP \in [0,1,...,63] \tag{4-14}$$

Sbits controls the precision of the $scale(i,j)$. Put (4-13) and (4-14) into (4-11):

$$F_q(i,j) = sign(G(i,j)) \times \left\lfloor \frac{\left(round\left(G(i,j) \times scale(i,j)\right) >> Sbit\right) \times Q_{tab}(QP) + Q_{offset} \times 2^{15}}{} \right\rfloor >> 15. \tag{4-15}$$

(4-15) is modified before implementation due to the following 3 reasons:

1. One can notice that the magnitude of the value in $\lfloor \cdot \rfloor$ in (4-15) is significantly less than 2^{15} such that it is most likely rounded to zero. To prevent this round off, a factor is added and it is removed after the inverse transform process.

2. When the input data block **X** is in *b*-bit (including sign bit), after order-16 DCT, its transform coefficients are in (*b*+4)-bit. The factor is chosen such that the quantized coefficient can be represented in (*b*+4)-bit with the minimum Q_{step} (i.e. 1). This makes sure the PIT coefficient has the same dynamic range as the DCT under the same quantization.

3. Since the transform process and quantization process in AVS are restricted to 16 -bit, bit truncation may required in the transform process. Furthermore, to maximize the usage of the 16-bit representation in the intermediate states, the bit shifting is adjusted. And finally, (4-15) becomes:

$$F_q(i,j) = sign(G(i,j)) \times \left\lfloor \frac{\left(round\left(G(i,j) \times scale(i,j)\right) >> Qbits\right) \times Q_{tab}(QP) + Q_{offset} \times 2^{18}}{} \right\rfloor >> 18. \quad (4\text{-}16)$$

where *Qbits* depends on the transform type. For example, it is 12 for LLMICT-A1.

4.4.2 Rescaling

When the quantized transform coefficient, which is computed in (4-16) with quantization step size Q_{step}, is received, they are rescaled to $\mathbf{F}_r$:

$$\begin{aligned} \mathbf{F}_r &= \mathbf{F}_q \, Q_{step} \\ &\approx \left(\mathbf{F}_q \cdot DQ_{tab}(QP\%8)\right) >> DQbits(QP) \end{aligned} \quad (4\text{-}17)$$

Here, $DQ_{tab}(QP)$ and *DQbits* are

$$DQ_{tab}(QP) = round\left(2^{15+\frac{QP\%8}{8}}\right). \quad (4\text{-}18)$$

$$DQbits(QP) = 12 - \left\lfloor \frac{QP}{8} \right\rfloor. \quad (4\text{-}19)$$

Finally, the output $\mathbf{Y}$ is reconstructed by inverse transform and the factor mentioned at the end of section 4.4.1 is removed by right shift by *Rbits*.

$$\mathbf{Y} = \left(\mathbf{E}^T \mathbf{F}_r \mathbf{E}\right) >> Rbits. \quad (4\text{-}20)$$

4.4.3 Example

It may not be easy to understand the pre-scaled transform with quantization and rescaling with formulae only. In order to display a clear picture of this process, here a step-by-step example will be shown.

Suppose the order-16 transform to be implemented is LLMICT-A1 with integer

kernel $\mathbf{E} = \mathbf{E}_{LLM\text{-}A1}$. Using (4-13) and $S_{bit} = 52$, the scaling matrix **SCALE** is:

$$\mathbf{SCALE} = \begin{bmatrix} 4096 & 4450 & 4745 & 4911 & 4520 & 4911 & 4745 & \cdots & 4745 & 4450 \\ 4450 & 4835 & 5155 & 5335 & 4911 & 5335 & 5155 & & 5155 & 4835 \\ 4745 & 5155 & 5496 & 5688 & 5236 & 5688 & 5496 & & 5496 & 5155 \\ 4911 & 5335 & 5688 & 5887 & 5419 & 5887 & 5688 & & 5688 & 5335 \\ 4520 & 4911 & 5236 & 5419 & 4987 & 5419 & 5236 & & 5236 & 4911 \\ 4911 & 5335 & 5688 & 5887 & 5419 & 5887 & 5688 & & 5688 & 5335 \\ 4745 & 5155 & 4742 & 5688 & 5236 & 5688 & 5496 & \cdots & 5496 & 5155 \\ \vdots & & & & & & \vdots & \ddots & \vdots & \vdots \\ 4745 & 5155 & 5496 & 5688 & 5236 & 5688 & 5496 & \cdots & 5496 & 5155 \\ 4450 & 4835 & 5155 & 5335 & 4911 & 5335 & 5155 & \cdots & 5155 & 4835 \end{bmatrix} \quad (4\text{-}21)$$

Assume QP = 34 such that it is not so high that all quantized coefficients become zero. Consider an intra-predicted input block **X** :

-2	-1	9	26	43	52	53	51	50	50	50	51	53	57	55	52
-3	10	36	42	40	43	42	41	39	37	39	40	40	44	45	42
16	24	19	-1	11	28	24	22	21	20	22	21	25	29	30	28
6	7	10	12	12	4	1	5	0	-2	6	11	10	11	13	12
-1	2	3	1	2	2	0	1	2	2	2	0	4	4	1	-4
-2	0	2	-3	-3	0	0	1	0	4	4	4	6	4	0	-10
0	0	-15	-8	3	-1	2	1	1	4	2	5	4	1	-8	-26
6	-31	-54	-10	3	-6	-3	1	-2	-4	2	8	4	-3	-1	0
-42	-53	-9	1	-5	-1	-1	-1	0	1	2	4	2	-4	-3	0
4	0	-1	-2	-4	-7	-4	1	0	-1	1	3	0	-3	2	6
-3	-3	-1	-4	-3	-2	-1	3	4	2	-14	-10	1	-2	0	1
1	0	1	-2	-1	1	1	-1	-3	-5	-11	-3	1	-2	-1	-2
1	-2	-2	-4	-4	-3	0	1	0	3	6	-1	-8	1	3	-1
1	1	-1	-3	-3	-1	0	0	1	4	3	-1	-2	2	3	0
3	2	-1	-3	-2	0	0	1	4	5	3	0	-1	-1	0	-3
1	-2	-3	-1	0	3	4	3	3	3	1	-3	-4	-5	-8	-4

It was captured from a real video sequence of 8-bit color depth. **G** is the intermediate result of the 2-D transform. Right shifts of 5 bits and 12 bits are applied in horizontal and vertical transform respectively to limit the intermediate results within 16-bit.

$$\mathbf{G} = \left(\mathbf{E}\left(\left(\mathbf{X}\,\mathbf{E}^{T}\right) >> 5\right)\right) >> 12 = \tag{4-22}$$

702	-339	-240	-88	-49	60	3	62	20	36	-4	22	2	31	5	-13
1114	-284	-104	-131	-130	37	-19	-7	-20	-3	-36	-6	15	9	-4	-4
1026	-59	-14	-94	22	-43	-31	-58	-130	-50	-27	-13	36	-7	5	0
580	-212	-56	-182	13	-57	-35	-41	10	-16	24	29	18	24	16	-10
266	-294	-297	-102	-72	5	76	59	110	64	41	16	-2	-13	-8	-9
164	-102	-144	26	-32	71	41	93	-8	52	-20	-17	-40	-29	-4	-9
120	95	-26	5	6	18	-74	4	-34	-67	6	-14	-26	6	4	-1
3	-112	-32	-30	-48	-110	-33	-30	-10	30	23	15	34	15	4	3
-161	-134	-88	-98	-58	-7	74	37	110	78	34	9	16	-12	-1	-5
-73	-2	47	27	60	113	67	109	99	-9	-26	-38	-68	-24	-18	-7
33	44	51	87	51	38	32	2	-33	6	-44	-37	-13	-16	7	2
-12	-43	-18	6	-35	8	-25	-49	-58	-33	-1	29	25	27	16	5
-58	-100	-29	-28	64	39	22	17	-10	-35	27	9	15	17	-3	-6
5	-23	53	94	149	93	104	53	12	-3	-31	-30	-19	-23	-21	-1
7	33	16	46	45	70	23	0	-36	-51	-17	-9	-10	-7	-2	6
-11	-24	-118	-31	-27	-35	-76	-10	-91	-46	12	19	26	24	19	8

Multiply with the scaling matrix:

$$\hat{\mathbf{G}} = \left(\mathbf{G} \odot \mathbf{SCALE}\right) >> 12 = \tag{4-23}$$

702	-368	-278	-106	-54	72	3	60	20	35	-5	26	2	37	6	-14
1210	-335	-131	-171	-156	48	-24	-7	-22	-3	-45	-8	18	12	-5	-5
1189	-74	-19	-131	28	-60	-42	-65	-151	-56	-36	-18	46	-10	7	0
695	-276	-78	-262	17	-82	-49	-47	12	-19	33	42	24	34	22	-13
294	-352	-380	-135	-88	7	97	63	121	68	52	21	-2	-17	-10	-11
197	-133	-200	37	-42	102	57	108	-10	60	-28	-24	-53	-42	-6	-12
139	120	-35	7	8	25	-99	4	-39	-75	8	-19	-33	8	5	-1
3	-118	-36	-35	-51	-127	-37	-28	-10	28	26	17	36	17	4	3
-161	-146	-102	-117	-64	-8	86	36	110	75	39	11	18	-14	-1	-5
-71	-2	53	31	64	131	75	102	96	-8	-29	-44	-72	-28	-20	-7
38	55	68	121	65	53	43	2	-38	7	-59	-51	-17	-22	9	3
-14	-56	-25	9	-46	11	-35	-57	-70	-38	-1	42	33	39	22	7
-64	-120	-37	-37	78	52	28	18	-11	-37	35	12	18	22	-4	-7
6	-30	74	135	197	134	144	61	14	-3	-43	-43	-25	-33	-29	-1
8	42	21	64	58	97	31	0	-42	-57	-23	-12	-13	-10	-3	8
-12	-28	-149	-40	-32	-46	-96	-10	-99	-48	15	25	31	31	24	9

Quantize the coefficient with $Q_{tab}(QP)$:

From (4-14) $$Q_{tab}(34) = round(2^{15-\frac{34}{8}}) = 1722. \quad (4\text{-}24)$$

$$F_q(i,j) = \left\lfloor \hat{G}(i,j) \times Q_{tab}(QP) + \left(Q_{offset} \times 2^{Qbits}\right) \right\rfloor >> Qbits = \quad (4\text{-}25)$$

4	-2	-2	-1	0	0	0	0	0	0	0	0	0	0	0	0
8	-2	-1	-1	-1	0	0	0	0	0	0	0	0	0	0	0
8	0	0	-1	0	0	0	0	-1	0	0	0	0	0	0	0
4	-2	0	-2	0	0	0	0	0	0	0	0	0	0	0	0
2	-2	-2	-1	0	0	0	0	1	0	0	0	0	0	0	0
1	-1	-1	0	0	0	0	1	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	-1	0	0	0	-1	0	0	0	0	0	0	0	0	0	0
-1	-1	0	-1	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0

These coefficients are zigzag scanned and sent then to the entropy coder. After lossless compression, these coefficients are sent to the decoder. When these coefficients are received by the decoder, they are rescaled to $\mathbf{F}_r$ first using (4-17):

From (4-18) $$DQ_{tab}(QP) = round\left(2^{15+\frac{34\%8}{8}}\right) = 38968. \quad (4\text{-}26)$$

From (4-19) $$DQbits(QP) = 12 - \left\lfloor \frac{34}{8} \right\rfloor = 8. \quad (4\text{-}27)$$

$$\mathbf{F}_v = \mathbf{F}_q \, Q_{step} \approx \left(\mathbf{F}_q \times DQ_{tab}(QP)\right) >> DQbits(QP) = \quad (4\text{-}28)$$

609	-304	-304	-152	0	0	0	0	0	0	0	0	0	0	0	0
1218	-304	-152	-152	-152	0	0	0	0	0	0	0	0	0	0	0
1218	0	0	-152	0	0	0	0	-152	0	0	0	0	0	0	0
609	-304	0	-304	0	0	0	0	0	0	0	0	0	0	0	0
304	-304	-304	-152	0	0	0	0	152	0	0	0	0	0	0	0
152	-152	-152	0	0	0	0	152	0	0	0	0	0	0	0	0
152	152	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	-152	0	0	0	-152	0	0	0	0	0	0	0	0	0	0
-152	-152	0	-152	0	0	0	0	152	0	0	0	0	0	0	0
0	0	0	0	0	152	0	0	0	0	0	0	0	0	0	0
0	0	0	152	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	-152	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	152	152	152	152	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	-152	0	0	0	0	0	0	0	0	0	0	0	0	0

Finally, after inverse 2-D transform, reconstructed Y is obtained by:

$$\mathbf{Y} = \left(\mathbf{E}^T \left(\left(\mathbf{F}_q \, \mathbf{E}\right) >> 8\right)\right) >> 15 = \quad (4\text{-}29)$$

1	1	12	30	44	48	50	49	51	45	44	46	50	53	55	58
2	15	29	37	41	45	44	36	36	36	34	30	34	41	44	40
15	18	15	10	15	27	31	22	17	23	27	25	23	25	24	21
1	9	14	11	7	7	8	5	2	2	6	11	12	12	10	8
-1	0	2	2	2	2	1	1	2	4	5	3	-1	-3	-1	1
0	1	1	0	-1	-3	-5	-3	4	10	8	3	-1	-1	-3	-7
-5	-9	-9	-4	-1	-1	0	4	7	6	3	1	1	-6	-15	-23
-11	-26	-30	-18	-6	-5	-1	5	3	-6	-5	5	7	-1	-4	3
-38	-35	-20	-4	-3	-11	-8	2	6	-4	-6	0	-1	-7	-2	10
-5	-5	-5	-6	-8	-8	-5	-2	-1	-1	1	1	-3	-7	-3	5
-4	-2	-3	-6	-4	2	5	2	-3	-5	-4	-3	-2	-1	-2	-3
-2	-2	-1	1	2	2	0	-3	-7	-12	-14	-11	-5	0	2	1
1	-4	-10	-10	-7	-4	-4	-4	0	3	0	-6	-7	-3	-1	-2
-10	-2	2	-1	-4	-3	-4	-5	2	8	4	-4	-2	6	7	0
9	7	1	-5	-3	7	10	6	2	5	3	-3	-6	-4	-4	-6
1	1	2	2	1	1	4	7	4	-1	-3	-1	-4	-9	-8	-3

4.5 Syntax Structures

Here are the syntax structures in our implementation. In intra block and inter block, the contents are indicated in different syntax structures.

4.5.1 Intra block

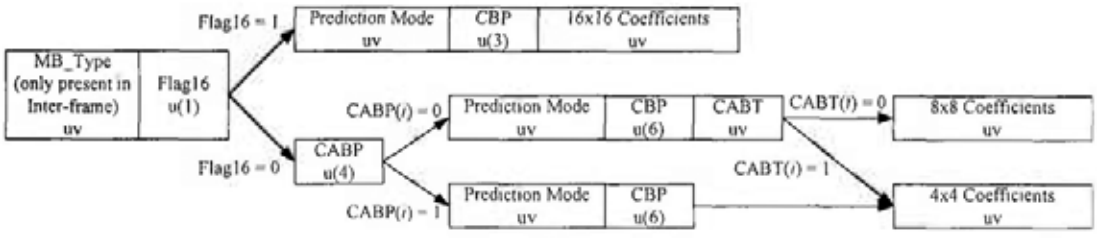


Figure 4-8 Intra-block syntax structure.

Figure 4-8 shows the syntax structure of an intra block. Each block is a syntax element. In the block, *uv* means it have variable bit length while *u*(*n*) means it has a fixed length of *n* bits.

Since MB in Inter frames can be inter-coded and also be intra-coded, an *MB_Type* header is required to indicate it. If the current frame is an intra frame, this *MB_Type* header is not needed. After that a 1-bit flag, *Flag16*, indicates if its residues are coded by order-16 transform. If *Flag16* = 1, the current MB is 16×16 intra-predicted and the residue is coded by an order-16 transform. Behind *Flag16* are the prediction mode, 3-bit Coded Bit Pattern (*CBP*, 1 bit for 16×16 luma block and 2 bits for two 8×8 chroma blocks). It indicates the present of non-zero coefficient in its corresponding block. Lastly, the non-zero transform coefficients are present. If *Flag16* = 0, smaller transforms are used. Since Combinative Arbitrary Block-size Prediction (*CABP*) is also proposed. It is possible that 4×4

and 8×8 intra predictions coexist in the same MB. A 4-bit *CABP* flag indicates the prediction size. Each bit correspond to an 8×8 sub-block. It is followed by the prediction modes and 6-bit *CBP* (4 bits for four 8×8 luma sub-blocks and 2 bits for two 8×8 chroma blocks). As 4×4 and 8×8 transforms can coexist in the same MB, a flag is needed to indicate each sub-block transform size. If *CABP* indicates the current sub-block i is 4×4 intra-predicted ($CABP(i) = 1$), the predicted residue must be coded with 4×4 transform and the MB syntax ends with transform coefficients. Otherwise, the current MB is 8×8 intra-predicted. Its residue can either be coded by 4×4 or 8×8 transform. This is indicated by a flag, *CABT*.

4.5.2 Inter block

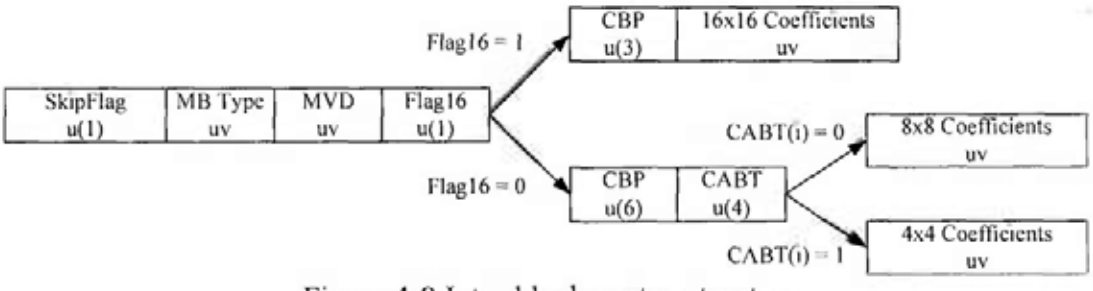


Figure 4-9 Inter-block syntax structure.

Figure 4-9 demonstrates the syntax structure of an inter-block. The notation is the same as intra block. Inter-block syntax structure is more straight-forward than intra-block. It first comes with a 1-bit *Skip* flag indicating if it is a skip block. If the current MB is not a skip block, it is followed by the *MB_Type* header which indicates the inter-prediction partition size. *MVD* stores the motion vector difference of each partition. A 1-bit *Flag16* indicates if the residue is coded with order-16 transform. If $Flag16 = 1$, the residue is coded with order-16 transform and the MB syntax ends with CBP and coefficients. If $Flag16 = 0$, CBP and CABT

indicating the use of either 4×4 or 8×8 transform in each sub-block. Lastly, it ends with the coefficients.

4.6 Entropy Coding

In AVS, the entropy coder can select either the Context-based Adaptive 2D Variable Length Coder (CA-2D-VLC) or the Enhanced Arithmetic Coder (EAC). In this thesis, the entropy coding adopted in the proposed platform is EAC. This is because it has a higher coding performance. In order to adapt to the order-16 transform coefficients, the EAC is modified. More context models are added to handle the long run of the transform coefficients. The structure of the EAC remains unchanged.

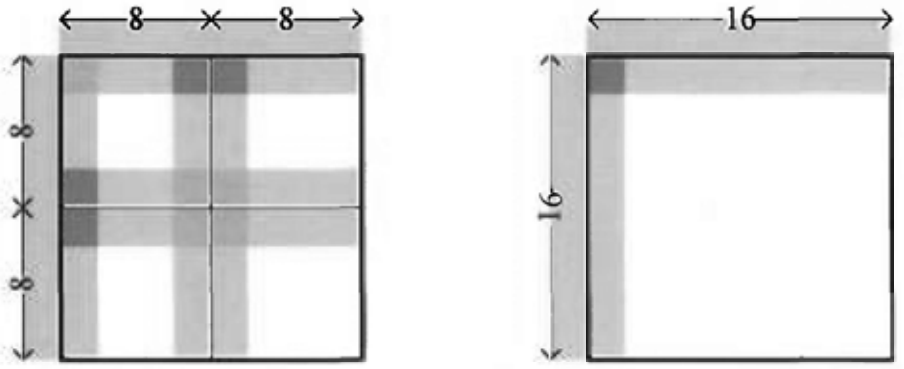
The context modeling of Flag16 and CBP16 are the same as those in H.264/AVC described in last chapter.

4.7 Loop Filter

In the original design of the loop filter in AVS, the loop filter is applied to the boundaries between the 8×8 sub-blocks. It is noticed that this design over-smooth some regions that are coded with order-16 transform. This over-smoothing effect may not be observed easily but lower the quality of the decoded frames (i.e. some of the reference frames). Hence, the qualities of the later frames may be affected.

To tackle this problem, we proposed to change the loop filter region when order-16 transform is used. The loop filter will only apply along the MB boundaries (16×16

block boundaries) when the MB is coded with order-16 transform. Those MB coded by order-4 and order-8 transform remain unchanged.



Region applying loop-filter in AVS standard

Region applying loop-filter if current MB is coded with order-16 Transform

Figure 4-10 Region of the loop filter is applied.

4.8 Experiment and Analysis

In our experiment, the testing platform is based on AVS reference software RM6.2*l*. The order-16 transforms stated in Section 4.3 are integrated to the testing platform. Implementation details are described in previous sections. This integrated platform is tested under the common test conditions specified by the AVS workgroup [18]. These testing conditions are listed in Table 4-2. Video sequences with resolutions from CIF (352×288) to HD (1920×1080) are tested. Some sequences, which are not specified in the common test conditions, are also tested.

In our analysis, the objective evaluation of different order-16 transforms will be shown first. The BD-bit rate and the equivalent BD-PSNR are measured using the method stated in [19]. These will show their RD performances. Their RD curves

will also be shown. After that, the subjective qualities will be compared. The decoded pictures will be shown.

Platform	RM6.2*l*	
Anchor	RM6.2*l* 4×4 and 8×8 transform enabled, 16×16 transform disabled.	
Prediction Structure	IBBPBBP…	
Intra Period	Every 1 second	
Frame Encoded	Sequence length is 4 seconds, at least 97 frames are encoded: WQVGA: 121~241 frames WVGA: 121~241 frames 720p: 241 frames 1080p: 97~202 frames	
Number of Reference Frames	2	
Entropy Coding	EAC (CABAC)	
ABT	Anchor: 4×4 and 8×8. Test: 4×4, 8×8 and 16×16.	
Rate Control	OFF	
RDO	ON	
Weighted Prediction	OFF	
Weighted Quantization	OFF	
Adaptive Interpolation Filter	OFF	
Loop Filter	ON (Jizhun profile)	
ME Search Range	32	
Target Bit Rates	Resolution	Bit Rate (kbps)
	WQVGA (416×240)	384, 512, 850, 1500
	WVGA (832×480)	512, 768, 1200, 2000
	720p	1600, 2500, 4000, 6000
	1080p@24Hz	1600, 2500, 4000, 6000
	1080p@50Hz	3200, 5000, 8000, 12000

Table 4-2 Testing conditions in AVS platform

4.8.1 RD Analysis (Objective Evaluation)

The experimental results are shown in Table 4-3 and Table 4-4. They display the BD-bit rate and BD-PSNR respectively. It can be easily observed that $\mathbf{T}_{Joshi}$ and LLMICT-A1 perform very similar. Their average bit rate reductions are over 8%. They are 8.26% and 8.20% respectively. They are equivalent to an average PSNR gain of 0.30 dB. The differences between them are within 0.06% but notice that LLMICT-A1 is simpler than $\mathbf{T}_{Joshi}$. It saves more than 10% computation time in the transform process with respect to $\mathbf{T}_{Joshi}$. MCSFICT and LMMICT-B1 are slightly lower than them. Their bit rate reductions are 7.46% and 7.32 (or 0.28dB and 0.27dB equivalent) respectively. The remaining including, MICT, $\mathbf{T}_{HI1}$, $\mathbf{T}_{HI2}$, $\mathbf{T}_{SI-AVS}$ and $\mathbf{T}_{Lee}$, all have average bit rate reduction below 7% (or 0.26 dB equivalent). The lowest is $\mathbf{T}_{Lee}$. It is 5.98% only (0.22 dB equivalent).

It can be noticed that order-16 transforms give better gain in HD sequences such as 720p and 1080p. The bit rate reductions are usually more than 10% for 1080p sequences. The largest gain is obtained by proposed LLMICT-A1 in "Crew" of 720p. A bit rate reduction of 15.89% (equivalent to 0.44 dB) is obtained. This is because it contains a large smooth region in the pictures and also many sudden intensity changes caused by flashes. These lead to a high usage of order-16 transform. This will be discussed in detail in later section. In the next section, the subjective quality of the decoded pictures will be compared.

BD-Bit Rate (%)		$T_{SI\text{-}AVS}$	MICT	T_{HI1}	T_{HI2}	T_{Lee}	T_{Joshi}	$T_{LLM\text{-}A1}$	$T_{LLM\text{-}B1}$	T_{MCSF}
CIF (352×288)	Foreman	-0.62	-1.03	-0.87	-0.92	-0.41	**-1.97**	-1.56	-1.23	-1.34
	Mobile	-0.38	-0.47	-0.54	-0.57	-0.40	-1.41	**-1.47**	-0.84	-1.09
	News	-0.19	-0.66	-0.95	-0.66	-0.78	**-1.58**	-1.23	-1.13	-1.48
	Paris	-0.25	-0.89	-1.33	-1.31	**-1.75**	-1.36	-1.03	-1.56	-1.71
	Average	-0.36	-0.76	-0.92	-0.87	-0.84	**-1.58**	-1.32	-1.19	-1.41
WQVGA (416×240)	BasketballPass	-4.88	-5.34	-5.05	-5.05	-3.17	**-6.39**	-6.25	-5.14	-5.74
	BlowingBubble	-6.32	-4.26	-6.74	-6.68	-5.70	-7.90	**-7.98**	-7.05	-7.50
	BQSquare	-9.01	-9.59	-9.66	-9.43	-9.67	-9.56	-9.41	-9.56	**-9.74**
	RaceHorses	-1.73	-0.42	-1.48	-1.34	-0.27	-2.39	**-2.85**	-1.60	-1.81
	Average	-5.49	-4.90	-5.73	-5.63	-4.70	-6.56	**-6.62**	-5.84	-6.20
WVGA (832×480)	BasketballDrill	-10.37	-10.76	-10.55	-10.53	-9.16	**-11.74**	-11.73	-10.84	-11.30
	BQMall	-8.00	-8.70	-8.85	-8.96	-8.71	-9.34	-9.16	-9.10	**-9.37**
	PartyScene	-10.76	-10.69	-10.58	-10.44	-9.53	**-11.11**	-10.95	-10.91	-10.95
	RaceHorses	-5.44	-5.71	-5.84	-5.55	-5.07	**-6.27**	-6.18	-5.71	-5.61
	Flowervase	-7.20	-7.14	-7.61	-7.57	-7.75	-7.67	-7.34	-7.42	**-7.93**
	Mobisode2	-14.05	-14.87	-14.65	-14.84	-13.61	-15.02	**-15.06**	-14.78	-14.81
	Keiba	-5.81	-6.98	**-7.37**	-6.92	-6.69	-6.40	-6.61	-6.84	-6.81
	Average	-8.80	-9.26	-9.32	-9.26	-8.65	**-9.65**	-9.58	-9.37	-9.54

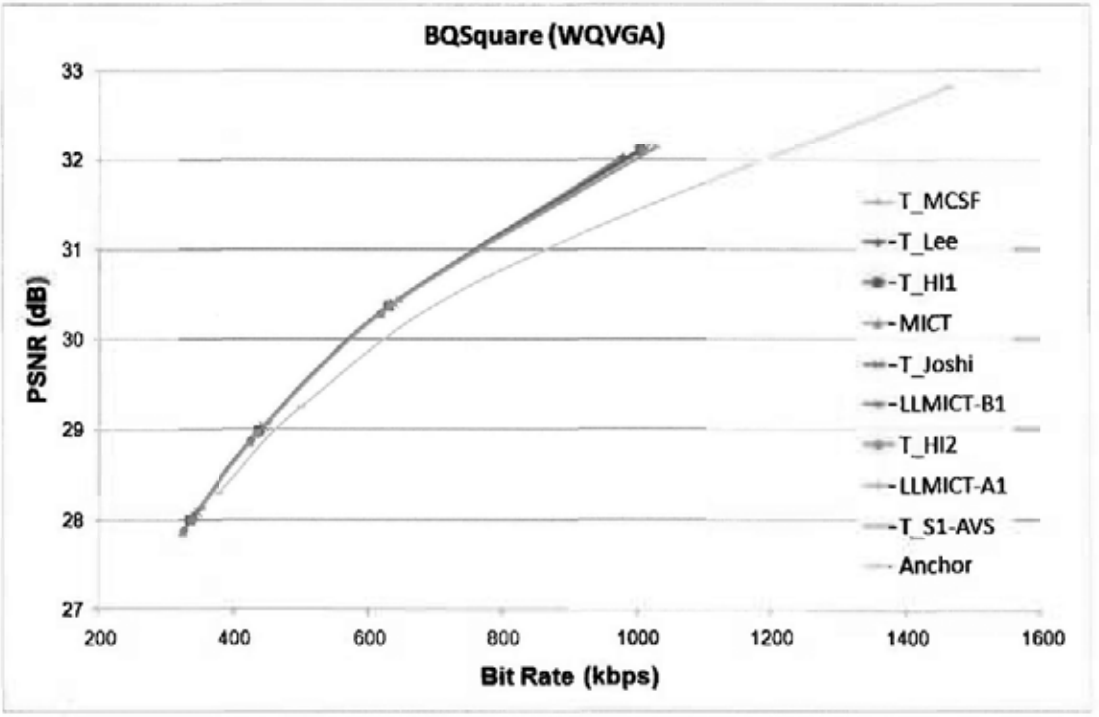
720p (1280×720)	Bigship	-2.61	-3.00	-2.86	-2.99	-1.67	**-3.68**	-3.58	-2.90	-3.05
	City	-5.48	-6.12	-5.97	-5.93	-5.22	-7.66	**-7.86**	-6.75	-6.33
	Crew	-14.44	-14.76	-14.64	-14.63	-13.29	-15.84	**-15.89**	-14.80	-14.93
	Harbour	-3.44	-4.73	-4.50	-4.53	-3.91	-6.93	**-6.96**	-5.68	-5.39
	Night	-1.92	-2.05	-0.81	-0.82	-0.62	-3.02	**-3.06**	-2.06	-2.22
	Raven	-0.97	-1.60	-0.69	-0.80	1.12	**-3.43**	**-3.43**	-1.37	-1.34
	Average	-4.81	-5.38	-4.91	-4.95	-3.93	-6.76	**-6.80**	-5.59	-5.54
1080p (1920×1080)	Kinomo1	-8.02	-10.02	-9.85	-9.63	-7.74	**-13.31**	-13.12	-10.70	-11.56
	Pedestrian	-10.47	-12.12	-12.18	-12.15	-11.05	**-14.12**	-13.94	-12.95	-13.11
	Sunflower	-7.18	-8.51	-7.06	-6.82	-3.75	-12.85	**-13.01**	-8.53	-7.29
	BasketballDrill	-10.32	-11.20	-11.65	-11.65	-10.84	**-12.69**	-12.42	-11.96	-12.44
	Cactus	-10.46	-11.04	-11.02	-11.00	-9.14	-13.23	**-13.39**	-11.68	-12.40
	RushHour	-10.74	-11.92	-11.81	-11.70	-10.48	**-13.49**	-13.21	-12.11	-11.79
	Station2	-9.75	-10.10	-9.59	-9.56	-8.21	**-10.93**	-10.79	-9.76	-9.89
	Average	-9.56	-10.70	-10.48	-10.36	-8.74	**-12.95**	-12.84	-11.10	-11.21
Average		-6.46	-6.95	-6.95	-6.89	-5.98	**-8.26**	-8.20	-7.32	-7.46

Table 4-3 Experimental Results of different transforms in AVS platform (BD-bitrate, %)

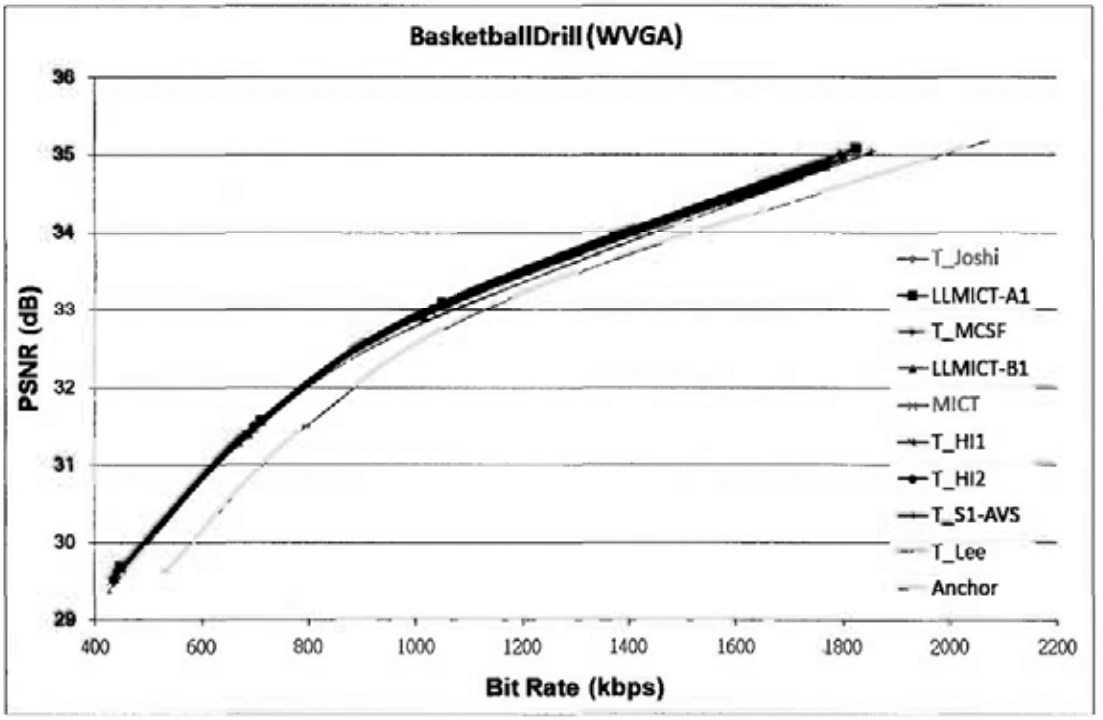
BD-PSNR (dB)		$T_{SI\text{-}AVS}$	MICT	T_{HI1}	T_{HI2}	T_{Lee}	T_{Joshi}	$T_{LLM\text{-}A1}$	$T_{LLM\text{-}B1}$	T_{MCSF}
CIF (352×288)	Foreman	0.02	0.04	0.03	0.04	0.02	**0.08**	0.07	0.05	0.05
	Mobile	0.02	0.02	0.02	0.02	0.01	0.05	**0.06**	0.03	0.04
	News	0.01	0.03	0.05	0.03	0.04	**0.08**	0.06	0.06	**0.08**
	Paris	0.01	0.04	0.07	0.06	**0.09**	0.07	0.05	0.08	0.08
	Average	0.02	0.03	0.04	0.04	0.04	**0.07**	0.06	0.06	0.06
WQVGA (416×240)	BasketballPass	0.22	0.24	0.22	0.23	0.14	**0.29**	0.28	0.23	0.26
	BlowingBubble	0.24	0.16	0.25	0.25	0.21	**0.30**	**0.30**	0.26	0.28
	BQSquare	0.32	0.34	0.34	0.34	0.34	0.34	0.33	0.34	**0.35**
	RaceHorses	0.08	0.02	0.06	0.06	0.01	0.10	**0.12**	0.07	0.08
	Average	0.22	0.19	0.22	0.22	0.18	**0.26**	**0.26**	0.23	0.24
WVGA (832×480)	BasketballDrill	0.41	0.43	0.42	0.42	0.36	**0.48**	**0.48**	0.44	0.46
	BQMall	0.35	0.38	0.39	0.39	0.38	**0.42**	0.41	0.37	**0.42**
	PartyScene	0.37	0.38	0.37	0.37	0.34	**0.39**	**0.39**	**0.39**	**0.39**
	RaceHorses	0.19	0.20	0.20	0.19	0.17	**0.22**	**0.22**	0.20	0.20
	Flowervase	0.35	0.35	0.37	0.37	0.37	0.37	0.36	0.36	**0.39**
	Mobisode2	0.63	0.67	0.66	0.67	0.61	**0.69**	0.68	0.67	0.67
	Keiba	0.31	0.37	**0.40**	0.37	0.35	0.34	0.36	0.37	0.37
	Average	0.37	0.40	0.40	0.40	0.37	**0.42**	0.41	0.40	0.41

720p (1280×720)	Bigship	0.07	0.08	0.08	0.08	0.05	**0.10**	**0.10**	0.08	0.08
	City	0.16	0.18	0.18	0.18	0.16	0.23	**0.24**	0.21	0.19
	Crew	0.39	0.41	0.40	0.40	0.36	**0.44**	**0.44**	0.41	0.41
	Harbour	0.13	0.18	0.17	0.17	0.14	**0.26**	**0.26**	0.21	0.20
	Night	0.07	0.07	0.06	0.06	0.02	**0.11**	**0.11**	0.07	0.08
	Raven	0.04	0.06	0.03	0.03	-0.04	**0.13**	**0.13**	0.05	0.05
	Average	0.14	0.16	0.15	0.15	0.12	**0.21**	**0.21**	0.17	0.17
1080p (1920×1080)	Cactus	0.28	0.30	0.30	0.30	0.25	0.36	**0.37**	0.32	0.34
	Sunflower	0.20	0.25	0.20	0.20	0.10	**0.39**	**0.39**	0.25	0.21
	BasketballDrive	0.33	0.36	0.37	0.37	0.35	**0.41**	0.40	0.39	0.40
	Kimono1	0.30	0.38	0.37	0.36	0.29	**0.52**	0.51	0.41	0.44
	Pedestrian	0.37	0.43	0.44	0.43	0.39	**0.52**	0.51	0.47	0.47
	Station2	0.34	0.36	0.34	0.34	0.29	**0.39**	0.38	0.34	0.35
	RushHour	0.34	0.38	0.38	0.38	0.33	**0.44**	0.43	0.39	0.38
	Average	0.31	0.35	0.34	0.34	0.29	**0.43**	**0.43**	0.37	0.37
Average		0.23	0.25	0.26	0.25	0.22	**0.30**	**0.30**	0.27	0.28

Table 4-4 Experimental Results of different transforms in AVS platform (BD-PSNR, dB)

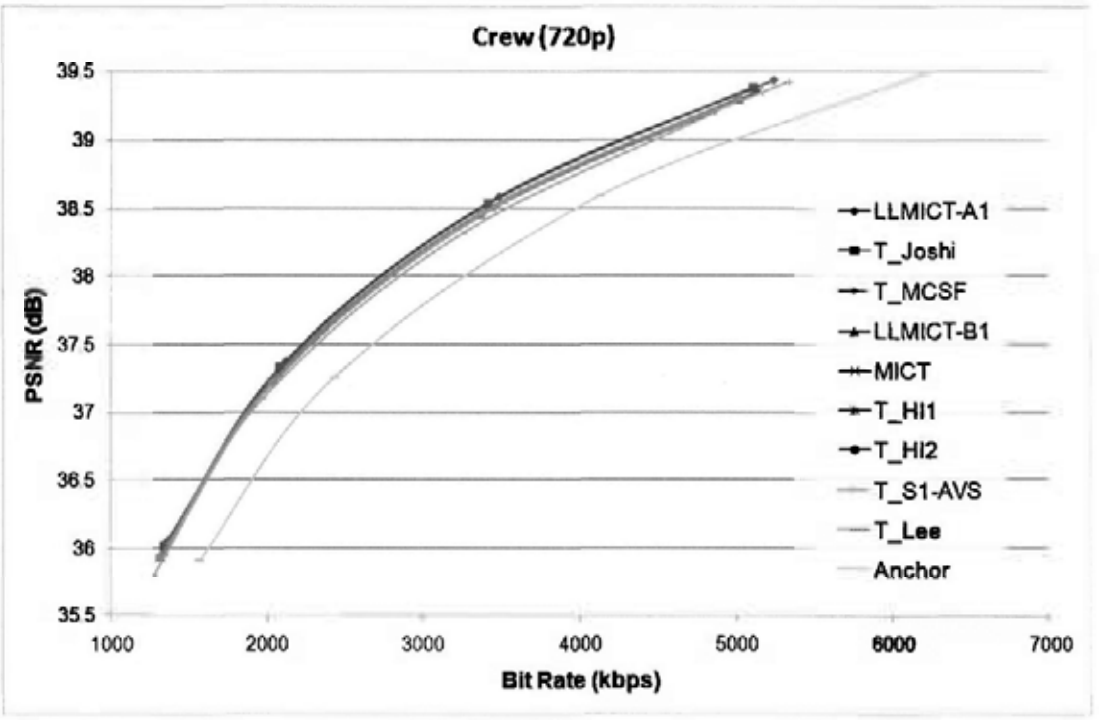


(a)

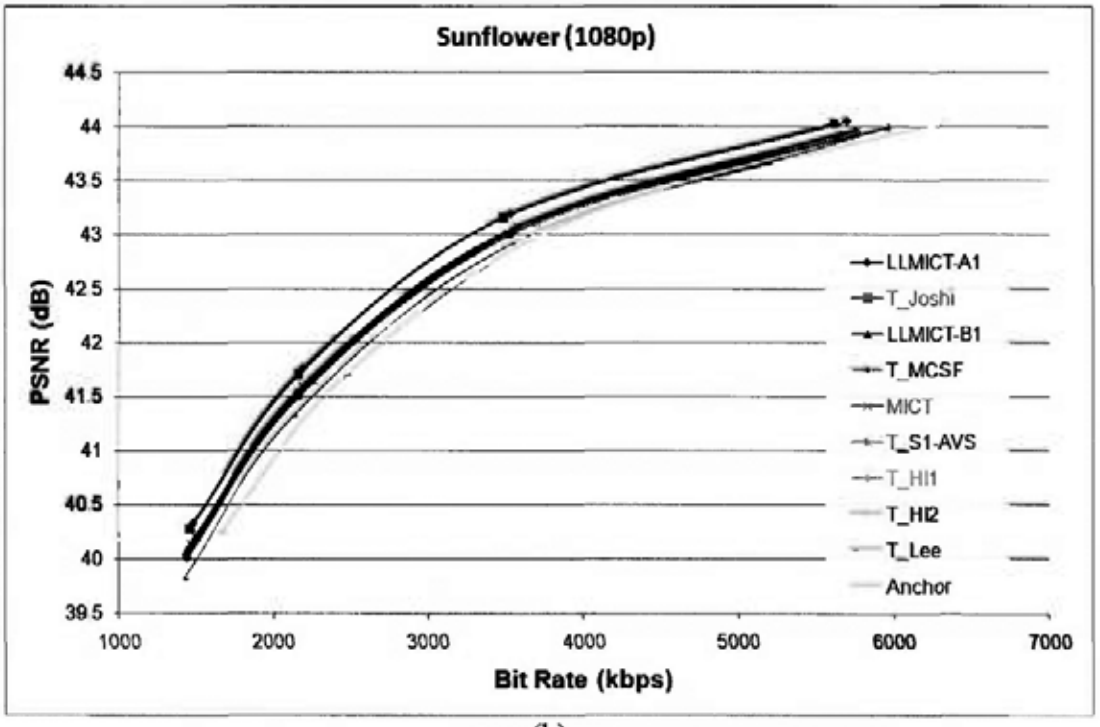


(b)

Figure 4-11 RD curves for (a) BQSquare (WQVGA) and (b) BasketballDrill (WVGA)



(a)



(b)

Figure 4-12 RD curves for (a) Crew (720p) and (b) Sunflower (1080p)

4.8.2 Subjective Evaluation

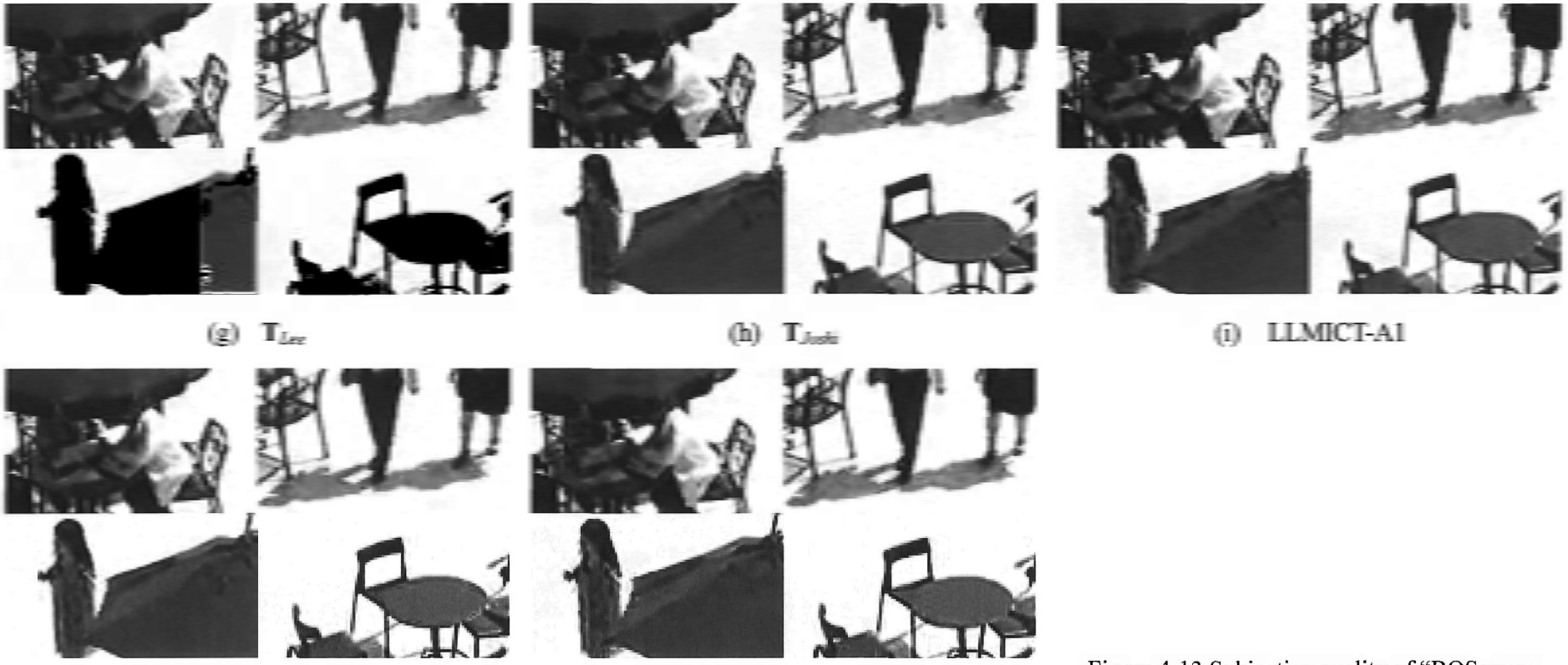
In this section, the subjective qualities of the decoded picture of different sequences will be shown. They have different resolutions and coded with different transforms.

The differences of subjective quality are not significant in lower resolutions such as WQVGA and WVGA. However, they are still observable. "BQSqaure" is a scene with high contrast. It is very obvious that ringing artifacts locates around the objects boundaries. This is significant in the anchor. When 16×16 transform is enabled, these artifacts are reduced. "BasketballDrill" contains regular local motion. Slight blocking artifacts are observed in Anchor. Again, these artifacts are reduced when 16×16 transform is used. In HD sequences, the distortions can be observed more easily. In "Crew", obvious blocking artifacts are observed in many smooth regions. The order-16 transforms dramatically suppress these artifacts.

However, the differences among different order-16 transforms are not significant. They can hardly be distinguished from their picture quality.

(a) Original (b) Anchor (c) $\mathbf{T}_{S1\text{-}AVS}$

(d) MICT (e) $\mathbf{T}_{HI1}$ (f) $\mathbf{T}_{HI2}$



(g) T_{Lee} (h) T_{Joshi} (i) LLMICT-A1

(j) LLMICT-B1 (k) MCSFICT

Figure 4-13 Subjective quality of "BQSquare (WQVGA)", 28th frame, coded at QP = 45.

(a) Original

(b) Anchor

(c) $\mathbf{T}_{SI\text{-}AVS}$

(d) MICT

(e) $\mathbf{T}_{HI1}$

(f) $\mathbf{T}_{HI2}$

(g) T_{Lee}

(h) T_{Joshi}

(i) LLMICT-A1

(j) LLMICT-B1

(k) MCSFICT

Figure 4-14 Subjective quality of "BasketballDrill (WVGA)", 20th frame, coded at QP = 37.

(a) Original

(b) Anchor

(c) $\mathbf{T}_{SI\text{-}AVS}$

(d) MICT

(e) $\mathbf{T}_{HI1}$

(f) $\mathbf{T}_{HI2}$

(g) $\mathbf{T}_{Lee}$

(h) $\mathbf{T}_{Joshi}$

(i) LLMICT-A1

(j) LLMICT-B1

(k) MCSFICT

Figure 4-15 Subjective quality of "Crew (720p)", 62nd frame, coded at QP = 34.

4.8.3 Usage of order-16 Transform

The average usages of order-16 transform (LLMICT-A1) are shown in Figure 4-16. Let us discuss in 4 different aspects.

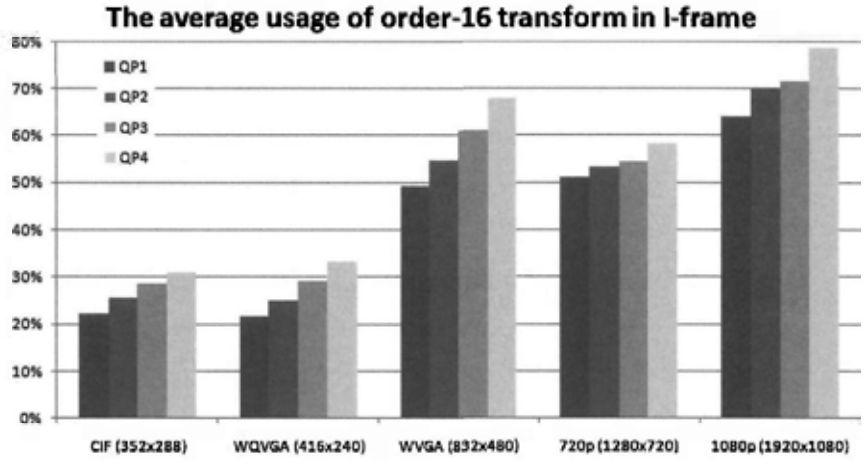
- **Frame Type:** It is shown that the usage of order-16 transform in AVS platform is pretty high in different frame types. They are 20% to 80% in I-frame, 20% to 50% in both P-frame and B-frame. The usage in I-frame depends on the picture nature. When a video sequence contains larger smooth region, the usage of order-16 transform will be higher. For example, "Flowervase" has an average usage over 70% in I-frame while "PartyScene" is only around 20%.

- **QP:** In our experiment, QP1 < QP2 < QP3 < QP4. We can notice that as QP increases, there is an obvious increasing trend in I-frame while there is a decreasing tread in B-frame. The trend in P-frame is not that obvious. Sequences with different resolutions have different trends as QP increases. The trend is changing from increasing to decreasing as the resolution increases.

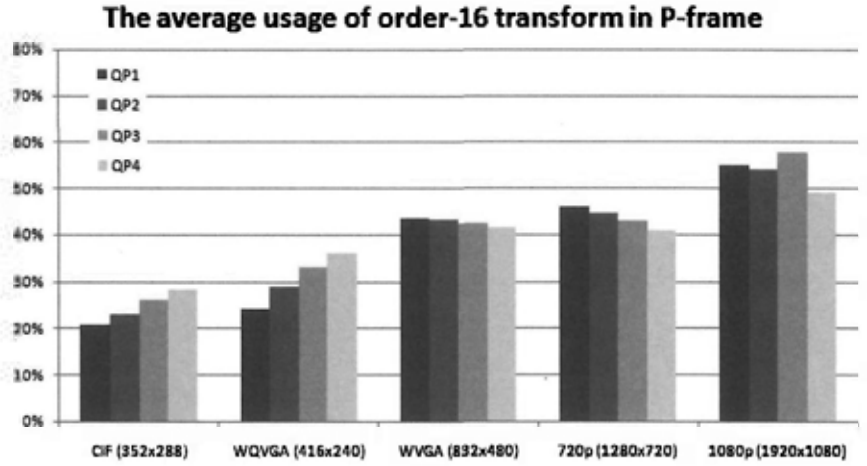
- **Resolution:** There is an increasing trend in all different frame types as the resolution increases. It is more obvious in I-frame that the usage increases from around 20% to almost 80%. It is not obvious in B-frame but the trend is still observable.

- **Vs. H.264/AVC**: One may compare the statistics in this AVS platform with the one obtained in H.264/AVC platform in last chapter. It is shown that the usage in I-frame in AVS is much higher than in H.264/AVC, especially for HD sequences. This is because the usage of intra 16×16 prediction in these

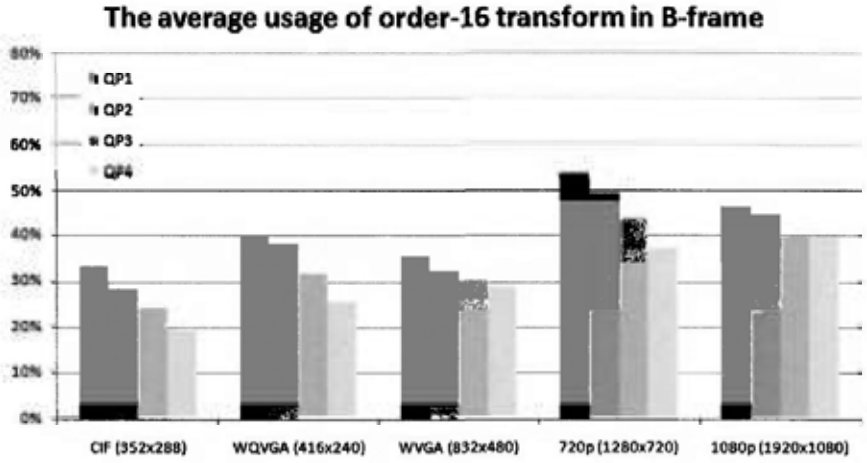
sequences is high. However, the transform selection in the MB with this prediction is compulsory to order-16 transform in AVS. It can be either order-4 or order-16 transform in H.264/AVC. Thus, the usage in AVS is higher than H.246/AVC. It can also be noticed that, in AVS platform, the usage is lower in P-frame while it is higher in B-frame (with respect to H.264/AVC). This is because they use different picture structures. H.264/AVC uses Hierarchical B (Hir-B) Structure (IbBbPbBbP…) while AVS uses IBBPBBP… structure. The P-frame period in H.264/AVC is longer than AVS such that the predicted residue is usually larger in the P-frame in H.264/AVC. Thus, less skip mode is used and hence the usage of order-16 transform is relatively higher in H.264/AVC. In contrast, the Hir-B structure reduces the predicted residue in B-frame (comparing with IBBPBBP… structure) such that the usage of skip mode in these frames is large. This implies a lower usage of order-16 transform in B-frames with Hir-B structure.



(a)



(b)



(c)

Figure 4-16 The average usage of order-16 in (a) I-frame, (b) P-frame and (c) B-frame.

4.9 Conclusions

In this chapter, we proposed an ABT platform for AVS coding standard as described in Section 4.2 to Section 4.7. Proposed order-16 integer transforms discussed in the last chapter are integrated into this platform and tested. They are compared with other existing order-16 transforms. Experimental results show that a significant gain is obtained when order-16 transform is used in video coding especially in HD sequences. The proposed transform, LLMICT-A1, gives a maximum bit rate reduction of 15.89% (equivalent 0.44 dB) in "Crew" sequence. On average, it offers a bit rate reduction of 8.2% (equivalent 0.30 dB). Other proposed transforms such as LLMICT-B1, MCSFICT, $\mathbf{T}_{S1\text{-}AVS}$, MICT, $\mathbf{T}_{HI1}$ and $\mathbf{T}_{HI2}$ offer average bit rate reductions over 6.4% (equivalent 0.23 dB). These transforms not only improve the objective coding performance but also the subjective quality. It is shown that the picture quality is improved when these transform is used. The blocking artifacts are significantly reduced.

We can conclude that our proposed ABT platform with proposed order-16 transforms significantly improves both the objective and subjective coding performance of AVS.

4.10 References

[1] GB/T20090.2 information technology — advanced audio video coding standard Part 2: Video, 2006.

[2] Fan Liang, Siwei Ma and Feng Wu, "*Overview of AVS video standard,*" International Conference on Multimedia and Expo, vol. 1, pp 423-426, 2004.

[3] Lu Yu, Feng Yi, Jie Dong and Cixun Zhang, "*Overview of AVS-Video: tools, performance and complexity,*" Visual Communications and Image Processing, Proc. of SPIE vol. 5960, pp 679-690, July 2006.

[4] Lu Yu, Sijia Chen and Jianpeng Wang, "*Overview of AVS-video coding standards,*" Signal Processing: Image Communication, vol. 24, issue 4, pp 247-262, April 2009.

[5] "*Next Generation AVS Video Coding Specification Version 2.0,*" AVS-N1590, March 2009. [Chinese]

[6] "*Next Generation AVS Video Coding - Call for Proposal,*" AVS-N1591, March 2009. [Chinese]

[7] C. Zhang, L. Yu, J. Lou, W. K. Cham and J. Dong, "*The Technique of Prescaled Integer Transform: Concept, Design and Applications,*" IEEE Trans on CASVT, vol. 18, no. 1, pp 84-97, 2008.

[8] Q. Wang, D. B. Zhao and W. Gao, "*Context-Based 2D-VLC Entropy coder in AVS Video Coding Standard,*" Journal of Computer Science and Technology, vol. 21, no. 3, pp 315-322, May 2006.

[9] W. K. Cham, C. K. Fong, Jie Dong, K. N. Ngan, H. M. Wong, Lu Wang, Yan Huo, Thomas Pun, "*Adaptive Block-size Transform for AVS-X,*" AVS Document AVS-M2284, 2008, [Chinese].

[10] W. K. Cham, C. K. Fong, Y. L. Fong, K. N. Ngan, Y. Liu and Carmen Cheng, "*Adaptive Block-size Transform towards AVS 2.0,*" AVS Document AVS-M2610, September 2009.

[11] C. K. Fong, W. K. Cham, Y. Liu and K. M. Cheng, "*Adaptive Block-size Transform towards AVS 2.0,*" AVS Document AVS-M2647, December 2009.

[12] W. K. Cham, C. K. Fong, Y. Liu and K. M. Cheng, "*An Investigation of Order-16*

Transform in AVS-M2606 ABT," AVS Document AVS-M2657, December 2009.

[13] C. K. Fong, W. K. Cham, Y. Liu and K. M. Cheng, "*Adaptive Block-size Transform towards AVS 2.0,*" AVS Document AVS-M2666, March 2010.

[14] Jie Dong, K. N. Ngan and W. K. Cham, "*Adaptive Block-size Transform for AVS X-profile,*" AVS Document AVS-M1771, March 2006.

[15] Yunfei Wang, Xunan Mao, Zhongmou Wu, Yun He, "AVS ABT Coding Technical Proposal," AVS Document AVS-M2303, 2008. [Chinese]

[16] Xunan Mao, Yunfei Wang, Yun He, W. K. Cham, C. K. Fong, Jie Dong, K. N. Ngan, H. M. Wong, Lu Wang, Yan Huo, Thomas Pun and Carmen Cheng, "*Adaptive block size coding for AVS-X profile,*" AVS Document AVS-M2372, 2008. [Chinese]

[17] Jay Loomis and Mike Wasson, "*VC-1 Technical Overview,*" October 2007. [Online] Available: http://www.microsoft.com/windows/windowsmedia/howto/articles/vc1techoverview.aspx

[18] "*AVS2 Common Test Conditions (Draft),*" AVS Document AVS-N1670, Dec. 2009. [Chinese]

[19] G. Bjøntegaard, "*Calculation of Average PSNR Differences between RD-curves,*" ITU-T SG16/Q6, Document VCEG-M33, April 2001. [Online] Available: http://wftp3.itu.int/av-arch/video-site/0104_Aus/

Chapter 5 Transform Domain Pattern Matching

5.1 Introduction

Pattern matching is a fundamental process in many image processing and computer vision applications. It involves matching a given image pattern to a target image by means of evaluating the similarity (or difference) between them. Suppose a $k \times k$ pattern $\mathbf{p}$ is to be matched with a windowed target image $\mathbf{w}$ of the same dimension. Both $\mathbf{p}$ and $\mathbf{w}$ are defined in a space $\boldsymbol{R}^{k \times k}$. The difference $\mathbf{d}$ between them is also in the same space.

$$\mathbf{d} = \mathbf{p} - \mathbf{w} . \tag{5.1}$$

The best candidate $\mathbf{w}_{best}$ is usually denoted as the candidate that gives the minimum $\mathbf{d}$ for all possible $\mathbf{w}$ in the candidate pool $\mathbf{W}$.

$$\mathbf{w}_{best} = \arg\min_{\mathbf{w}\in\mathbf{W}} \mathbf{d}(\mathbf{p},\mathbf{w}). \tag{5.2}$$

To quantify the difference **d**, different measures are used. For example sum of absolute difference (SAD) and sum of square difference (SSD) are commonly used.

$$d_{SAD}(\mathbf{p},\mathbf{w}) = \sum_{i=0}^{k-1}\sum_{j=0}^{k-1}\left|p(i,j)-w(i,j)\right|. \tag{5.3}$$

$$d_{SSD}(\mathbf{p},\mathbf{w}) = \sum_{i=0}^{k-1}\sum_{j=0}^{k-1}\left|p(i,j)-w(i,j)\right|^2. \tag{5.4}$$

The choice of measure is application dependent. No matter which measure is used, the matching process is a very computation intensive work. Many fast pattern matching methods have been proposed to speed up the pattern match process and to maintain the accuracy at the same time. These fast pattern matching algorithms can be classified into two main classes. The first one is reducing the complexity of the similarity evaluation. Every possible candidates in the target image are evaluated but with simplified evaluation metric. One of the typical examples is Fast Full Search (FFS). The accuracy of this class of algorithms is the same as Exhaustive Full Search. No degradation is observed. As every candidate is examined, however, the computation load is still very high. In contrast, another class of the fast algorithms targets to reduce the size of the candidate pool **W**. Only portion of the candidates is examined. A number of fast motion estimation methods [1]-[3] are in this class of pattern matching, such as Three-step search and Diamond search. These algorithms are significantly simpler than fast full search. However, there is a trade-off between this speed-up and the accuracy. As only some but not all candidates are examined, the best candidate may not be examined and hence, the accuracy is lowered.

5.2 Pattern Matching in Walsh-Hadamard Domain

Hel-Or [7]-[9] proposed a high speed pattern matching algorithm for noisy images. This algorithm projects the pattern and the windowed target image into the 2-D Walsh Hadamard (WH) domain. The Euclidean distance between the projected pattern patch and the projected target image patch are evaluated. Mismatched patters are eliminated in an early stage. It is fast but its performance is approaching to the full search. The search result is in pixel accuracy. It can be a preliminary result for sub-pixel search. Mak and Li proposed a motion estimation method using Hel-Or's fast pattern matching algorithm.

In projection-based pattern matching, $\mathbf{p}$ and $\mathbf{w}$ projected onto the m^{th} WH basis (denoted as $\mathbf{u}_m$) are denoted as $b_p(m)$ and $b_w(m)$ respectively. Using Cauchy-Schwartz inequality:

$$\|\mathbf{u}_m\| \|\mathbf{d}\| \geq \left\|\mathbf{u}_m^T \mathbf{d}\right\|. \tag{5.5}$$

Consider the Euclidean difference:

$$\begin{aligned} d_E(\mathbf{p},\mathbf{w}) &= \|\mathbf{p}-\mathbf{w}\| = \|\mathbf{d}\| \\ &\geq \frac{\left\|\mathbf{u}_m^T(\mathbf{p}-\mathbf{w})\right\|}{\|\mathbf{u}_m\|} = \frac{\left\|\mathbf{u}_m^T\mathbf{p}-\mathbf{u}_m^T\mathbf{w}\right\|}{\|\mathbf{u}_m\|}. \end{aligned} \tag{5.6}$$

When there is a collection of $\mathbf{u}_m$ such that $\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, \ldots \mathbf{u}_m]$ and the corresponding project collection $\mathbf{b} = [b_1, b_2, \ldots b_m]$, (5.6) can be expressed as:

$$d_{SSD}(\mathbf{p},\mathbf{w}) = \|\mathbf{d}\|^2 \geq \mathbf{b}^T\left(\mathbf{U}^T\mathbf{U}\right)^{-1}\mathbf{b}. \qquad (5.7)$$

This is the distance lower bound for a set of projections vectors. This lower bound is getting tighter and tighter when the number of projections increases. As a result, the pattern match can be performed in WH domain in an iterative manner. After each projection, the candidates whose lower bound value is greater than a threshold are rejected. The lower bound values of the remaining candidates are updated before next projection. This is a recursive process until a predefined number of best matches are found or the maximum number of projection is reached. WH basis vectors have strong energy packing ability such that most of the energy is packed in the first few projections. As a result, the comparison can be terminated in first few projections. Majority of the mismatch candidates are removed from the candidate pool quickly and hence less computation is required.

Besides the high speed of this algorithm, this algorithm also has an accuracy same as full search. It also has a very high robustness. It is not affected by the difference in illumination and noisy environment.

Based on the above Hel-Or's idea, Li and Mak proposed a faster pattern matching in Walsh-Hadamard transform domain called Fast Walsh Search (FWS) [10]. In this fast pattern matching, two techniques, Block Pyramid Matching and Partial Sum of Absolute Difference, are proposed.

5.2.1 Block Pyramid Matching

Block Pyramid Matching (BPM) is an algorithm to compute the WH projections in a hierarchical structure such that intermediate result can be reused. As a result, several projections can be generated at the same time. A $2^n \times 2^n$ block can be divided into four $2^{n-1} \times 2^{n-1}$ blocks. The DC or the $(0, 0)^{th}$ projection of the $2^n \times 2^n$ block can be decomposed into the sum of the $(0, 0)^{th}$ projections of the four $2^{n-1} \times 2^{n-1}$ sub-blocks. The $(0, 1)^{th}$, the $(1, 0)^{th}$ and the $(1, 1)^{th}$ projections of the $2^n \times 2^n$ block can be decomposed into combinations of the $(0, 0)^{th}$ projections of the four $2^{n-1} \times 2^{n-1}$ sub-blocks in the same manner. Recursively, they can be decomposed into combinations of the $(0, 0)^{th}$ projections of 2×2 sub-blocks. In Figure 5-1, an example of BPM is shown. The DC of the 8×8 block can be decomposed into the sum of DC of the 4×4 sub-blocks. The DC of each 4×4 sub-block can be further decomposed into sum of DC of the four 2×2 sub-blocks. When the pattern sliding window slides by 2 pixels, the DC of each 2×2 sub-blocks can be reused. As a result, the intermediate results are shared among different sliding window positions and hence the computation can be significantly reduced.

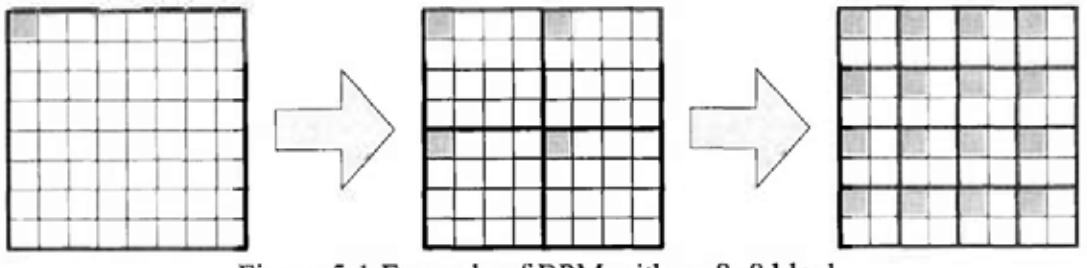


Figure 5-1 Example of BPM with an 8×8 block.

5.2.2 Partial Sum of Absolute Difference

In Hel-Or's algorithm, the projected distance is measured by the Euclidean distance which involves square operations or multiplications. In Li and Mak's algorithm, it is suggested to be replaced by absolute difference. It is called the partial sum of absolute difference (PSAD, or partial absolute difference in [10]). If $\Phi(\mathbf{p}, \mathbf{w}; q)$ is the PSAD of q projections between $\mathbf{p}$ and $\mathbf{w}$, $\Phi(\mathbf{p}, \mathbf{w}; q)$ is given as:

$$\Phi(\mathbf{p},\mathbf{w};q) = \Phi(\mathbf{p},\mathbf{w};q-1) + \frac{\left|\mathbf{u}_q^T\mathbf{p} - \mathbf{u}_q^T\mathbf{w}\right|}{\left\|\mathbf{u}_q\right\|_1} \quad \text{for } q > 1 \text{ and } \Phi(\mathbf{p},\mathbf{w};0) = 0. \qquad (5.8)$$

After each projection, PSAD of the candidates in the pool are calculated. If one's PSAD is larger than a threshold T_0, this candidate will be removed from the pool. It is in a recursive manner until the number of candidates in the pool is less than a preset threshold or the maximum projection is reached. The use of PSAD makes the algorithm multiplication-free and closer to metrics in the codec which measure the sum of absolute difference (SAD) instead of SSD. However, the threshold T_0 for eliminating mismatch candidates is not easy to determine. Here we propose a statistical threshold and the Block Adaptive Threshold.

5.2.3 Statistical Threshold

It is obvious that the best match candidate may not always have the minimum PSAD. Suppose the PSAD for the best match candidate and minimum PSAD in the candidate pool are Φ^{BM} and Φ^{min} respectively. There is a real value T such that

$$\Phi^{BM} = \Phi^{\min} + T. \tag{5.9}$$

For all candidates, there is a relationship that:

$$\Phi = \Phi^{\min} + t \tag{5.10}$$

where t is a real value random variable. The probability that $t > T$

$$\begin{aligned} P(t > T) &= P(\Phi^{\min} + t > \Phi^{\min} + T) \\ &= P(\Phi > \Phi^{BM}) \end{aligned} \tag{5.11}$$

When the probability density function (pdf) of t, $p_t(t)$, is known, (5.11) becomes:

$$\begin{aligned} P(\Phi > \Phi^{BM}) &= P(t > T) \\ &= \int_{-\infty}^{T} p_t(\tau) d\tau. \end{aligned} \tag{5.12}$$

Although $p_t(t)$ cannot be found analytically yet, it can be found empirically. It is shown in Figure 5-2. If we can tolerate a very low miss rate such that the best match may be removed from the candidate pool at a very low probability P_{miss}, we can find a value T_0 such that:

$$\int_{-\infty}^{T_0} p_t(\tau) d\tau \geq 1 - P_{miss}. \tag{5.13}$$

The minimum value of T_0 that satisfies (5.13) is the threshold to eliminate the mismatch candidates.

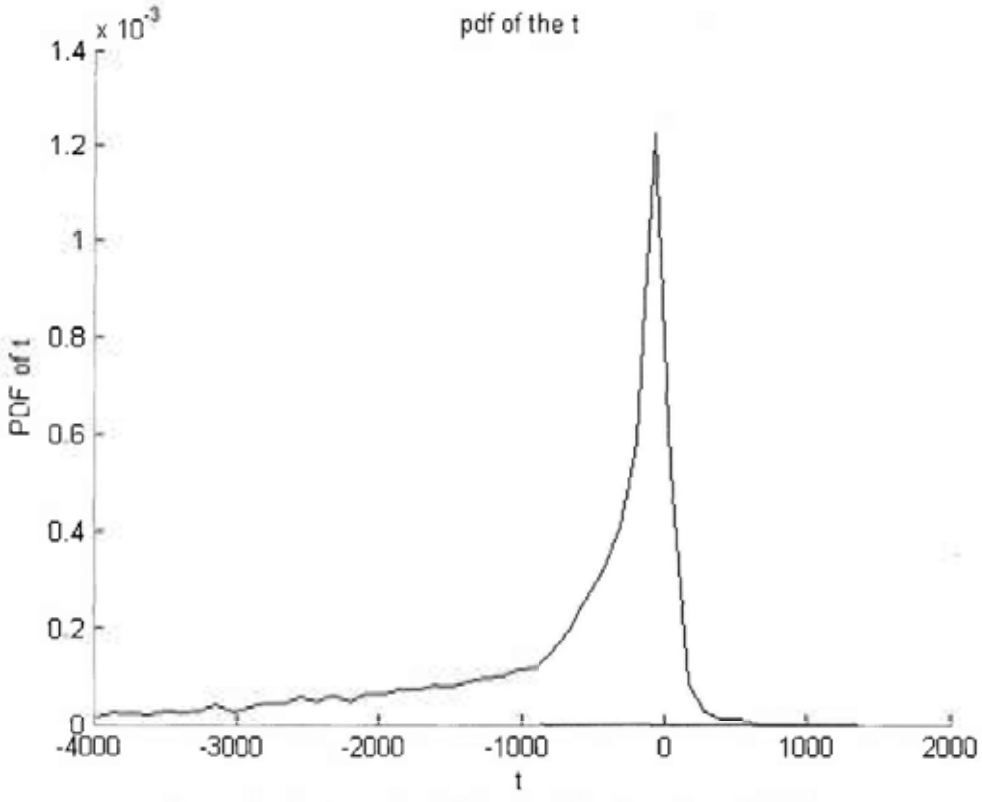


Figure 5-2 The probability density function of $p_t(t)$.

In block-based motion estimation, the candidates are in the search range of $(\pm R, \pm R)$. Mak found that the candidate located at (0, 0) is very likely to have $\Phi = \Phi^{\min}$. Let Φ of this candidate be Φ_0, (5.9) and (5.10) become:

$$\Phi^{BM} = \Phi_0 + T, \tag{5.14}$$

$$\Phi = \Phi_0 + t. \tag{5.15}$$

Now, t is defined as the difference between Φ and Φ_0. Its pdf can also be found from a set of training sequences and the shape is similar to that in Figure 5-2. The threshold T_0 can be found with (5.13). Mak found that when T_0 is equal to 0 and Φ is obtained from the first two projects (i.e. $q = 2$ in (5.8)), the miss rate is acceptable in a balance between the computation and the accuracy. This threshold, $T_0 = 0$, is adopted in the fast motion estimation in [10].

5.2.4 Block Adaptive Threshold

The statistical threshold discussed in the last section is a fixed threshold at a fixed acceptable miss rate. This works very well for majority of the blocks. However, the fixed threshold may be over-estimated for very smooth blocks or under-estimated for highly-textured blocks. This leads to inefficient mismatch elimination for those very smooth blocks and over-elimination of the candidates for highly textured blocks. As a result, we proposed an adaptive threshold which depends on the nature of the pattern block **p**. For simplicity, the 8×8 pattern blocks are classified into two types: smooth blocks and texture blocks. Each block type has its own $p_t(t)$ and hence its own threshold T_0. This is called Block Adaptive Threshold. Using the definition in (5.15), the value of T_0 of these block types are found (Table 5-1). When the target miss rate is even as low as 30%, it is shown that affect of the RD performance is very minor. The number of the candidates for smooth block significantly reduces while the miss rate for texture block is lowered. Overall, the remaining candidates are reduced by 10%. However, the time for the matching process does not improve much (~1%) because the classification of the pattern block also takes time.

Target Miss Rate (%)	T_0	
	Texture Block	Smooth Block
1.0	280	40
5.0	73	9
10	16	1
20	-20	-3
30	-48	-6
40	-110	-32

Table 5-1 Block adaptive threshold of texture blocks and smooth blocks

5.3 Experiments

A Variable Block Size Motion Estimation (VBS-ME) algorithm based on FWS (denoted as FWS-VBS algorithm) is built [10]. This algorithm is integrated into the H.264/AVC reference software JM 10.1 [11]. FWS-VBS is compared with other VBS-ME algorithms implemented in the reference software. The VBS-ME to be compared including Fast Full Search (FFS), hybrid UMHexagons (UMH) [12], simplified UMHexagons (SUMH) and Enhanced Predictive Zonal Search (EPZS) [13]. Video sequences with frame resolution from CIF (352×288) to HD (1280×720) are tested. The GOP structure is IPPPP… is used. Only the first frame is intra-coded. QP = {16, 20, 24 and 28} are tested. The BD-bitrates described in [14] are measured. They are shown in Table 5-2. The best case in each sequence is bolded. It is obvious that the test VBS-ME algorithms have similar RD performance as FFS. The average BD-bitrates are less than 0.5%. In comparison, EPZS performs the best. It is followed by UMH, FWS-VBS and SUMH. However, without professional optimization, FWS-VBS is just slower than SUMH. It only requires 20% of the time for FFS on average. In [10], an early mode stop (EMS) is proposed such that the required computation time is almost halved.

Resolution	Sequences	BD-bit rate compare with FFS (%)			
		UMH	SUMH	EPZS	FWS-VBS
CIF (352×288)	Akiyo	**0.05**	0.40	0.24	0.15
	Coastguard	-0.78	**-1.31**	-1.00	-0.52
	Container	**0.03**	0.11	0.18	**0.03**
	Foreman	0.80	2.84	0.60	**0.30**
	Mother	0.40	1.09	0.69	**0.28**
	News	0.12	0.73	**0.09**	0.38
	Tempete	-0.53	-0.42	**-0.56**	-0.31
SIF (352×240)	Mobile	-1.07	-0.70	**-1.14**	-0.73
	Stefan	-1.79	-1.87	**-2.08**	-1.34
SD (704×576)	City	0.60	0.48	0.23	**0.00**
	Crew	0.03	0.00	**-0.18**	0.16
	Harbour	-1.25	**-2.30**	-1.46	-0.95
	Ice	1.45	2.44	**0.92**	1.26
	Soccer	-0.46	**-0.64**	-0.47	-0.21
HD (1280×720)	Mobile-Cal	-0.26	-0.26	**-0.33**	-0.26
	Night	-0.26	-0.09	**-0.55**	2.23
	Panslow	0.91	3.22	0.78	**0.39**
	Raven	**-0.04**	1.29	0.46	0.23
	Sailormen	-0.05	0.04	**-0.25**	0.04
	Shuttle-Start	**0.51**	1.30	1.07	0.61
Average		-0.08	0.32	**-0.14**	0.09

Table 5-2 BD-bit rates of different VBS-ME algorithms

Resolution	Sequences	Percentage of ME time compare with FFS			
		UMH	SUMH	EPZS	FWS-VBS
CIF (352×288)	Akiyo	12.9	9.5	13.6	11.8
	Coastguard	31.1	20.0	27.4	28.4
	Container	15.4	10.2	16.3	13.2
	Foreman	22.8	13.1	21.0	19.5
	Mother	14.0	10.1	15.5	12.8
	News	13.9	10.1	15.9	13.0
	Tempete	28.3	16.8	24.9	23.8
SIF (352×240)	Mobile	31.0	20.0	27.2	27.1
	Stefan	26.3	16.9	24.6	24.8
SD (704×576)	City	28.8	17.8	25.9	26.9
	Crew	29.8	15.2	23.9	23.5
	Harbour	32.0	18.7	27.5	29.5
	Ice	14.6	10.2	14.9	13.1
	Soccer	25.4	15.0	23.3	22.7
HD (1280×720)	Mobile-Cal	27.7	16.2	23.7	23.0
	Night	26.0	13.9	22.8	24.1
	Panslow	22.1	11.9	18.3	17.9
	Raven	17.8	11.5	18.6	16.0
	Sailormen	29.4	16.4	24.4	26.5
	Shuttle-Start	12.6	8.7	12.5	10.8
Average		23.1	14.1	21.1	20.4

Table 5-3 Computation time of different VBS-ME algorithms

5.4 Conclusions

In this chapter, a transform domain pattern matching method called Fast Walsh Search (FWS) is described. It was proposed by Li and Mak based on Hel-Or's work. The matching is in Walsh-Hadamard transform domain in which pixel energy in the patch are packed into several transform coefficients. Less comparison is required and hence the matching process is speeded up. To further increase the speed of the process, Block Pyramid Matching (BPM) and Partial Sum of Absolute Difference (PSAD) were suggested. One of our contributions to this work is derivation of a statistical threshold for eliminating the mismatch candidates. This threshold can be found by an empirical pdf when tolerance of the missing the best match is specified. With this threshold, majority of the candidates are removed from the candidate pool while the best match is kept. This effectively speeds up the matching process. Another contribution is the investigation of the Block Adaptive Threshold. Different blocks have different nature. Smooth blocks and textured blocks should have different thresholds. Experiment shows that the threshold values for these two kinds of block have a big difference. Although the Block Adaptive Threshold cannot give a significant speed-up to the process, it shows a possible direction to do so. The FWS was integrated to H.264/AVC reference JM10.1. It is compare with some state-of-the-art motion estimation algorithms such as UMH, SUMH and EPZS. The accuracies of these motion estimation algorithms are similar but the proposed FWS is just slower than the fastest algorithm SUMH. FWS only takes 20% of the computation time for the Fast Full Search.

5.5 References

[1] T. Koga, K. Iinuma, A. Hirano, Y. Iijima and T. Ishiguro, "*Motion compensated interframe coding for video conferencing*," in Proc. Nat. Telecommunication Conference, pp. G5.3.1-5.3.5, 1981.

[2] L. M. Po and W. C. Ma, "*A novel four-step search algorithm for fast block motion estimation*," IEEE Trans. on CASVT, vol. 6, no. 3, pp. 313-317, June 1996.

[3] S. Zhu and K. K. Ma, "*A new diamond search algorithm for fast block-matching motion estimation*," IEEE Trans. on Image Processing, vol. 9, no. 2, pp. 287-290, Feb. 2000.

[4] Chun-Man Mak, Chi-Keung Fong and Wai-Kuen Cham, "*Fast Motion Estimation for H.264/AVC in Walsh Hadamard Domain*," IEEE Trans. on CASVT, vol. 18, no. 6, pp. 735-745, June 2008.

[5] C. M. Mak, N. Li and W. K. Cham, "*Fast motion estimation in Walsh Hadamard domain,*" Proceeding of International Symposium on Intelligent Signal Processing and Communication Systems, pp. 349-352, 2005.

[6] N. Li and W. K. Cham, "*Statistical threshold for real time pattern matching using projection kernels,*" Proceeding of International Symposium on Intelligent Signal Processing and Communication Systems, pp. 57-60, 2005.

[7] Y. Hel-Or, H. Hel-Or, "*Real time pattern matching using projection kernels,*" Proceeding of 9th IEEE International Conference on Computer Vision, Vol. 1, pp. 1486-1493, Oct. 2003.

[8] Y. Hel-Or, H. Hel-Or, "*Real time pattern matching using projection kernels,*" IEEE Trans on PAMI, Vol. 27, no. 9, pp. 1430-1445, 2005.

[9] M. Ben-Yehuda, L. Cadany and H. Hel-Or, "*Irregular Pattern Matching using Projections,*" IEEE International Conference on Image Processing, Vol. 2, pp. 834-837, 2005.

[10] N. Li, C. M. Mak and W. K. Cham, "*Fast block matching algorithm in Walsh-Hadamard domain,*" in Proc. Asian Conference of Computer Vision (ACCV), pp. 712-721, Jan. 2006.

[11] H.264/AVC Reference Software JM10.1 [Online]. Available:

http://iphome.hhi.de/suehring/tml/download/old_jm

[12] Z. Chen, J. Xu, Y. He and J. Zheng, "*Fast integer-pel and fractional-pel motion estimation for H.264/AVC,*" Journal of Visual communication and Image Representation, vol. 17, no. 2, pp. 264-290, Apr. 2006.

[13] A. M. Tourapis, O. C. Au and M. L. Liou, "*Highly efficient predictive zonal algorithms for fast block-matching motion estimation,*" IEEE Trans. on CASVT, vol. 12, no. 10, pp. 934-947, Oct. 2002.

[14] G. Bjøntegaard, "*Calculation of Average PSNR Differences between RD-curves,*" ITU-T SG16/Q6, Document VCEG-M33, April 2001. [Online] Available: http://wftp3.itu.int/av-arch/video-site/0104_Aus/

Chapter 6 Distribution Modeling of Predicted Residue Transform Coefficient

6.1 Introduction

A number of works has been done in the analysis of the transform coefficient (usually refer as DCT coefficient) distribution of images [1]-[7]. They are commonly expected to be in Laplace distribution. As a result, it is assumed that the DCT coefficients are in Laplace distribution in many applications such as image coding. In fact, the transform coefficient distribution is a fundamental to many applications, such as rate control, quantization noise analysis and transform-based restoration. An accurate distribution model is very important to these applications. Beside Laplace distribution, the transform coefficients have been proposed to be modeled as mixture of Gaussian and Generalized Gamma Distribution.

In video coding, predicted residue is transformed instead of the pixel data. Therefore, the distribution of the transform coefficient of predicted residue should be focused. It is commonly expected to be in Laplace distribution also. It is recently found that, however, the predicted residue is closer to Cauchy distribution than Laplace distribution. In [8] and [9], the transform coefficient is modeled in Cauchy distribution and more accurate rate and distortion models are achieved.

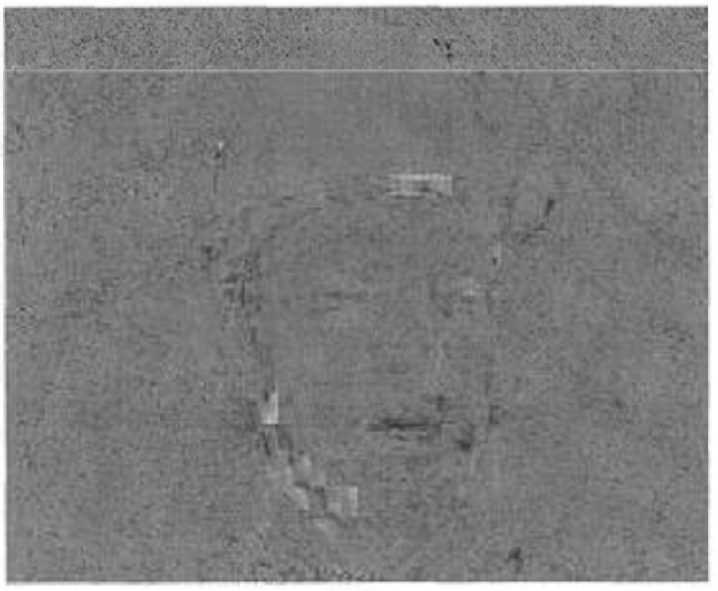
Usually, the parameters of the distribution are estimated from the transform coefficients. In this chapter, the parameters of the Cauchy distributed transform coefficient are going to be estimated from the predicted residue without being transformed. First of all, the predicted residue is verified to be Cauchy distributed. After that, some properties of Cauchy Distribution will be described. Using these properties, the distributions of the transform coefficient will be derived. Experiments will verify our proposed model.

6.2 Distribution of Predicted Residue

In video coding, prediction is used to reduce the temporal and the spatial redundancy in pixel data. Intra-prediction reduces the spatial redundancy while inter-prediction reduces the temporal one. The predicted residue is sent after being transformed and quantized. In general, the residue from intra-prediction has a larger variance than that from inter-prediction. Figure 6-2 shows the predicted residue of the video sequence, Foreman. The upper is the intra-predicted while the lower is the inter-predicted. Their histograms are plotted in Figure 6-2. It is shown that the intra-predicted residue has a wider spread (i.e. larger variance) than the inter-predicted. It can be shown that

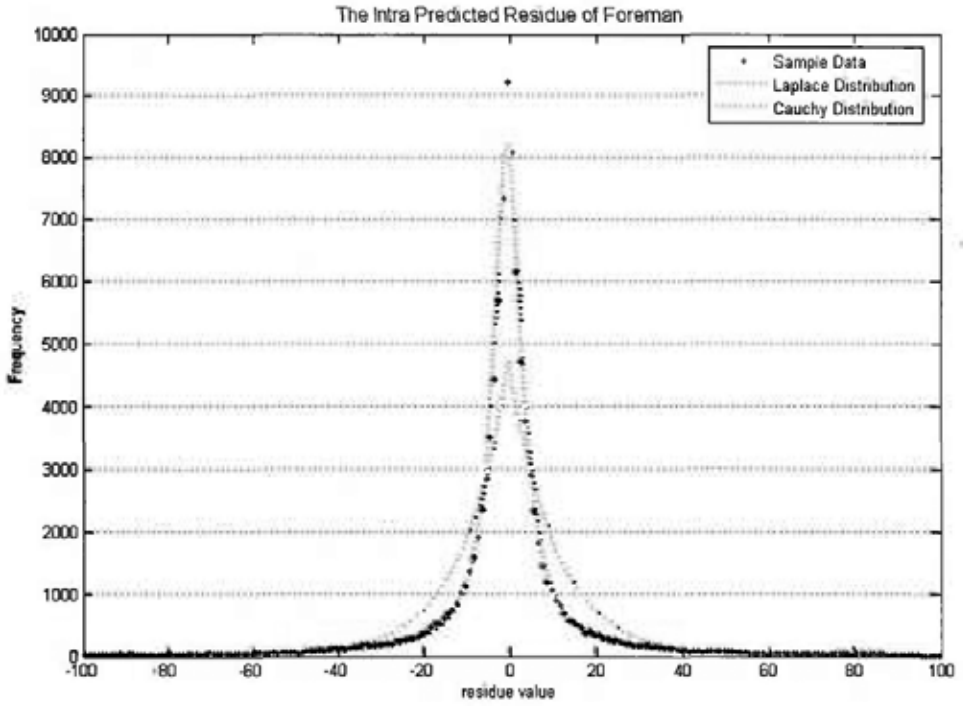
the distribution of the intra predicted residue is closer to Cauchy distribution while the inter predicted is closer to Laplace distribution.

(a)

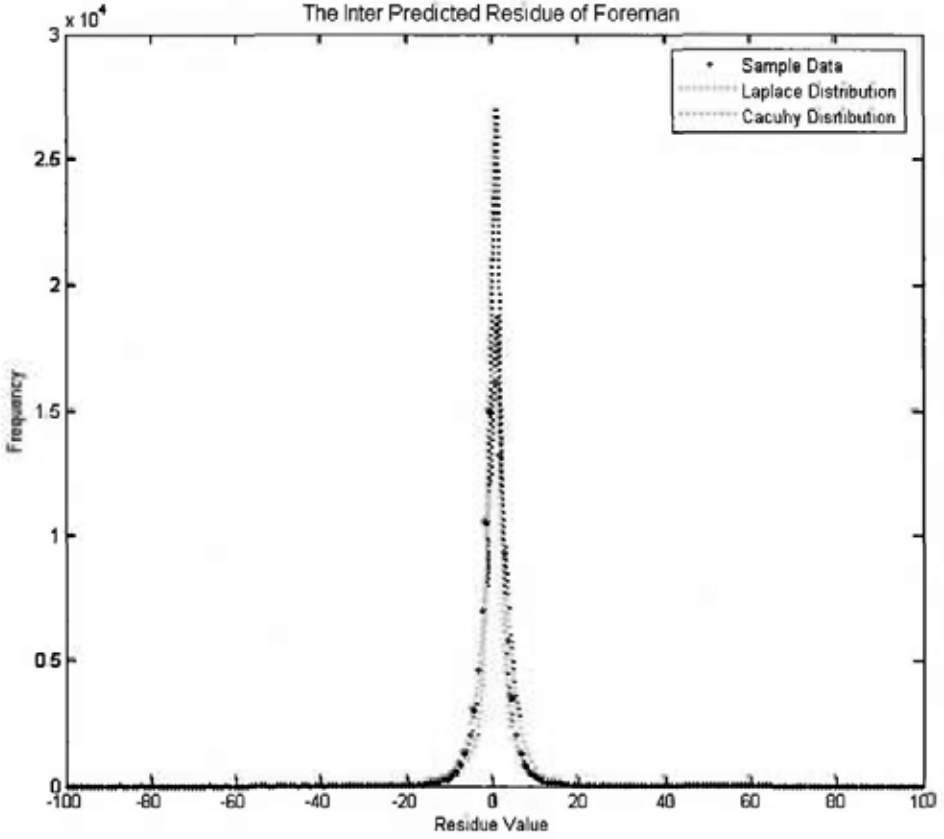


(b)

Figure 6-1 (a) The intra-predicted residue and (b) the inter-predicted residue of "Foreman".



(a)



(b)

Figure 6-2 The distribution of (a) the intra-predicted residue and (b) the inter-predicted residue of "Foreman".

6.3 Properties of Laplace Distribution

Laplace distribution is described by the probability density function (pdf):

$$p(x;\mu,b) = \frac{1}{2b}\exp\left(-\frac{|x-\mu|}{b}\right). \tag{6.1}$$

Its mean and variance is given by μ and $2b^2$ respectively. Its parameters are commonly estimated by the maximum likelihood (ML) estimator [10]. Their estimated values are:

$\hat{\mu}$ is the median of the samples, and (6.2)

$$\hat{b} = \frac{1}{N}\sum_{i=1}^{N}|x_i - \hat{\mu}|. \tag{6.3}$$

6.4 Properties of Cauchy Distribution

Cauchy distribution can be described by the pdf:

$$p(x;\mu,\gamma) = \frac{\gamma}{\pi\left((x-x_0)^2+\gamma^2\right)}. \tag{6.4}$$

In this pdf, x_0 is the location parameter, which is the median and the mode of the distribution and γ is the scale parameter, which describe the spread of the distribution. The standard Cauchy random variable (RV), X, is defined as:

$$X \sim C(0,1) = \frac{1}{\pi\left(x^2+1\right)}. \tag{6.5}$$

The Cauchy RV can be expressed as:

$$\frac{\gamma}{\pi\left((x-x_0)^2+\gamma^2\right)} = C(x_0,\gamma) = \gamma \cdot C(0,1) + x_0 . \tag{6.6}$$

Cauchy distribution is an example of a more generalized version of the central limit theorem. It is a stable distribution such that the sum of independent Cauchy distributed RV is also in Cauchy distribution:

$$\begin{aligned} &X_i \sim C(x_{0,i},\gamma_i), \\ &Y = \sum_i X_i \sim C\left(\sum_i x_{0,i}, \sum_i \gamma_i\right). \end{aligned} \tag{6.7}$$

However, there is no mean variance defined for Cauchy distribution. This makes the parameters not easy to be estimated.

6.1.1. Parameter Estimation Method 1

The simplest way to estimate the parameters is approximating its median by its ensemble mean:

$$\overline{X} = \frac{1}{N}\sum_{i=1}^{N} X(i) \approx x_0 . \tag{6.8}$$

And from (6.6), the peak of the pdf is:

$$p_{\max} = p(x = x_0) = \frac{1}{\pi\gamma}. \tag{6.9}$$

As a result, γ is estimated by:

$$\hat{\gamma} = \frac{1}{\pi \cdot p_{\max}}. \tag{6.10}$$

This method is very simple but not very robust. The accuracy is low when the samples are noisy or insufficient number of the observed data. The empirical pdf is required.

6.1.2. Parameter Estimation Method 2

Another method is by means of entropy. The entropy of a Cauchy distribution is:

$$H = \log_2(4\pi\gamma). \tag{6.11}$$

When the pdf or the histogram of the observed samples is known, the entropy of these observed samples in N bins is:

$$H_O = -\sum_{i=1}^{N} p_O(x_i)\log_2 p_O(x_i) \tag{6.12}$$

where $p_O(x_i)$ is observed probability density at x_i. Since these two entropies should be the same, hence:

$$\hat{\gamma} = \frac{2^{H_o}}{4\pi}. \tag{6.13}$$

This is a robust method with high accuracy. However, the empirical pdf is still needed and the entropy of the observed data has to be found. This requires more computation and memory.

6.1.3. Parameter Estimation Method 3

In video coding, the range of the predicted residue value is bounded. Let this bound be $[X_{\min}, X_{\max}]$ such that the Cauchy RV:

$$X_i \in [X_{\min}, X_{\max}], \forall X_i. \tag{6.14}$$

Assume that the distribution is symmetric about x_0 and:

$$R = X_{\max} - x_0 \approx x_0 - X_{\min} >> \gamma \tag{6.15}$$

and the approximation:

$$\tan^{-1}\phi \approx \frac{\pi}{2} - \frac{1}{\phi} \text{ when } \phi >> 1. \tag{6.16}$$

The mean and variance become of X_i:

$$\begin{aligned}
\overline{X}_i &= \int_{X_{\min}}^{X_{\max}} x \cdot p(x)dx \\
&= \frac{\gamma}{\pi}\left[\frac{1}{2}\ln\left(\frac{\gamma^2 + (x_0 - X_{\max})^2}{\gamma^2 + (x_0 - X_{\min})^2}\right)\right] \\
&\quad - \frac{x_0}{\pi}\left[\tan^{-1}\left(\frac{x_0 - X_{\max}}{\gamma}\right) - \tan^{-1}\left(\frac{x_0 - X_{\min}}{\gamma}\right)\right] \\
&\approx -\frac{x_0}{\pi}\left[\left(-\frac{\pi}{2} + \frac{\gamma}{R}\right) - \left(\frac{\pi}{2} - \frac{\gamma}{R}\right)\right] \\
&= -\frac{x_0}{\pi}\left(-\pi + \frac{2\gamma}{R}\right) \\
&= x_0\left(1 - \frac{2\gamma}{\pi R}\right) \\
&\approx x_0
\end{aligned} \tag{6.17}$$

$$\begin{aligned}\sigma_X^2 &= \int_{X_{min}}^{X_{max}} (x - x_0)^2 \cdot p(x)dx \\ &= \frac{\gamma}{\pi}\left[(X_{max} - X_{min}) + \gamma\left(\tan^{-1}\left(\frac{x_0 - X_{max}}{\gamma}\right) - \tan^{-1}\left(\frac{x_0 - X_{min}}{\gamma}\right)\right)\right] \\ &\approx \frac{\gamma}{\pi}\left[2R + \gamma\left(\left(-\frac{\pi}{2} + \frac{\gamma}{R}\right) - \left(\frac{\pi}{2} - \frac{\gamma}{R}\right)\right)\right] \\ &= \frac{\gamma}{\pi}\left[2R + \gamma\left(-\pi + \frac{2\gamma}{R}\right)\right] \\ &= \frac{2R\gamma}{\pi} - \gamma^2 + \frac{2\gamma^3}{\pi R}\end{aligned} \tag{6.18}$$

For Intra-predicted residue, its variance is relatively large and hence it can be assumed to be zero-mean. In Figure 6-2, the distribution is shown to be symmetric around zero

$$x_0 \approx 0 \tag{6.19}$$

and γ is the solution to:

$$\gamma^3 - \frac{\pi R}{2}\gamma^2 + R^2\gamma - \frac{\pi R\sigma_x^2}{2} = 0 \tag{6.20}$$

where

$$R \approx \frac{X_{max} - X_{min}}{2}. \tag{6.21}$$

As a result, the parameters of the Cauchy distribution can be estimated easily from the observed sample variance. The empirical pdf of the observed sample is not required. However, this method is sensitive to the accuracy of X_{max}. This significantly affects the robustness.

6.5 Transform Coefficient of Predicted Residue

In Section 6.2, it is shown that the predicted residue has a Cauchy distribution. Transform coefficient of predicted residue is a linear sum of the predicted residue. In (6.7), it is shown that linear sum of the Cauchy RV is also in Cauchy distribution. As a result, the distribution of the transform coefficients is a Cauchy distribution. The best way to estimate the distribution parameters of the transform coefficients is from the complete statistics of the transform coefficients. However, in many applications, these statistics are not available in a single pass. As a result, a two-pass or even a multi-pass strategy is used. Transformation is done and the statistics is gathered in the first pass while the process is done in the remaining passes. This is an accurate method but a very time-consuming one. Here, a method to estimate the distribution parameter of the transform coefficients without transforming the residue is proposed.

Image and video pixel data are usually modeled in first order Markov process:

$$x_i = \rho \cdot x_{i-1} + \varepsilon. \tag{6.22}$$

The predicted residue can also be modeled in the same way with zero-mean. In 1-D case, the autocorrelation matrix of the predicted residue x is:

$$\mathbf{R}_{xx} = \{\rho^{|i-j|}\}. \tag{6.23}$$

When x is transformed with an orthogonal kernel $\mathbf{T}$, the autocorrelation matrix of the transform coefficient is:

$$\mathbf{R}_{cc} = \mathbf{T}\,\mathbf{R}_{xx}\,\mathbf{T}^T. \tag{6.24}$$

If x has a variance of σ_x^2, the variance of the i-th transform coefficient $c(i)$ is:

$$\sigma_c^2(i) = \sigma_x^2 R_{xx}(i,i). \tag{6.25}$$

Assume that the 2-D case is separable into row-order and column-order 1-D process. The variance of the 2-D transform coefficient $c(r, s)$ is:

$$\sigma_c^2(r,s) = \sigma_x^2[\mathbf{T}\,\mathbf{R}_{xx}^{row}\,\mathbf{T}^T]_{(r,r)}[\mathbf{T}\,\mathbf{R}_{xx}^{col}\,\mathbf{T}^T]_{(s,s)} \tag{6.26}$$

where $\mathbf{R}_{xx}^{row}$ and $\mathbf{R}_{xx}^{col}$ are the 1-D autocorrelation matrices in row- and column-order respectively. They are defined as:

$$\mathbf{R}_{xx}^{row} = \{\rho_{row}^{|i-j|}\} \quad \text{and} \quad \mathbf{R}_{xx}^{col} = \{\rho_{col}^{|i-j|}\} \tag{6.27}$$

where ρ_{row} and ρ_{row} are the correlation coefficients in row- and column-order respectively. The parameters of transform coefficient distribution can be estimated by substituting σ_x^2 in (6.20) with $\sigma_c^2(r,s)$ in (6.26). This implies that the transform coefficient distributions can be found when the variance, the column- and row-order correlation are known.

6.6 Experimental Results

In this chapter, our transform coefficient distribution parameter estimation method will be evaluated. To evaluate it, the goodness-of-fit is measured and compared. There are a number of goodness-of-fit tests. One of them is the Chi-Square test (χ^2-test).

6.1.4. Chi-Square Test

The χ^2-test is to test the hypothetical distribution against the observed data. The cumulative distribution function (CDF) of the hypothetical distribution is assumed to be available as F. For the observed data classified in k bins such that the frequency of the observed samples in the i^{th} bin is O_i. The expected frequency in the i^{th} bin from the hypothetical distribution is:

$$\begin{aligned} E_i &= N \cdot [F(x_U) - F(x_L)] \\ &= N \cdot \int_{x_L}^{x_U} p(x)dx. \end{aligned} \tag{6.28}$$

N is the number of observed data, x_U and x_L are the upper and lower bound of the i-th bin. The test statistic is defined as:

$$\chi^2 = \sum_{i=1}^{k} \frac{(O_i - E_i)^2}{E_i}. \tag{6.29}$$

The smaller the value of χ^2, the better fit of the hypothetical distribution is. It can be noticed that χ^2 will increase as the number of observation, N, increases. In our cases, N depends on the video resolution. Here we normalize χ^2 by N:

$$\chi^2_{norm} = \frac{1}{N}\sum_{i=1}^{k}\frac{(O_i - E_i)^2}{E_i}. \qquad (6.30)$$

This does not affect the physical meaning of the χ^2 but makes the comparison among different resolutions easier.

6.1.5. Empirical Results

In our experiment, the predicted residue is transformed by the order-4 ICT adopted in H.264/AVC with normalization. The statistics is in frame-based. The histograms of the predicted residue and its transform coefficients are gathered every frame. Their distribution parameters are estimated with our proposed method. With these parameters, the hypothetical distributions are formed and the goodness-of-fit test is taken every frame. The average, minimum and maximum values of normalized χ^2 in each tested sequence will be listed and compared.

The result of fitting the intra-predicted residue is shown in Table 6-1. It is observed that the predicted residues in most tested sequences are closer to Cauchy distribution. It is also shown that Cauchy Estimator 1 (i.e. the parameter estimation method 1) is more accurate than Cauchy Estimator 2 and Cauchy Estimator 3 in our experiment. The evaluation results of the transform coefficient modeling are shown from Table 6-2 to Table 6-7. The results for the most important coefficients, the first 6 coefficients along the zig-zag scan path, are shown. These coefficients contribute over 80% of the signal energy. From the χ^2_{norm} value obtained from the ML Laplace Estimator and the Cauchy Estimator 1, it is shown that the distributions of the transform coefficients are closer to Cauchy distribution than Laplace distribution. Although the

proposed Cauchy Estimator is not as good as the Cauchy Estimator 1, they are still comparable.

6.7 Conclusions

In this chapter, the predicted residues of different sequences are investigated. It is shown that the intra-predicted residue is closer to the Cauchy distribution. Three Cauchy distribution parameter estimation methods are proposed and tested. The estimation method from the peak of the empirical pdf is found to be very accurate when the observed data are sufficient. Due to the properties of Cauchy distribution, it is expected that the transform coefficients of the intra-predicted residue are also Cauchy distributed. A method is proposed to estimate the Cauchy distribution parameters of the transform coefficients without transformation.

Parameter Estimator		Cauchy Estimator 1			Cauchy Estimator 2			Cauchy Estimator 3			ML Laplace Estimator		
GOF χ^2_{norm}		Max.	Ave.	Min.	Max.	Ave.	Min.	Max.	Ave.	Min.	Max.	Ave.	Min.
CIF	Foreman	0.026	0.017	0.013	0.037	0.022	0.016	0.083	0.086	0.051	0.219	0.196	0.180
	Football	0.011	0.005	0.002	0.036	0.024	0.010	0.052	0.031	0.012	0.077	0.048	0.032
	Paris	0.038	0.034	0.030	0.087	0.072	0.060	0.619	0.590	0.560	0.310	0.392	0.275
	Coastguard	0.049	0.029	0.020	0.172	0.151	0.129	0.274	0.192	0.112	0.014	0.010	0.005
	Mobile	0.104	0.090	0.074	0.091	0.081	0.073	0.650	0.604	0.573	0.269	0.48	0.227
WVGA	Flowervase	0.280	0.241	0.202	0.556	0.367	0.226	0.511	0.350	0.223	0.921	0.769	0.632
	Keiba	0.032	0.014	0.07	0.113	0.071	0.048	0.095	0.059	0.034	0.220	0.132	0.097
	Mobisode2	0.246	0.227	0.208	0.935	0.793	0.552	2.540	1.151	0.602	0.789	0.653	0.553
	RaceHorses	0.016	0.012	0.008	0.037	0.031	0.022	0.048	0.034	0.020	0.136	0.115	0.102
720p	City	0.117	0.101	0.089	0.128	0.116	0.107	0.132	0.118	0.107	0.273	0.262	0.241
	Crew	0.137	0.061	0.045	0.198	0.161	0.103	0.305	0.225	0.099	0.488	0.225	0.196
	Harbour	0.035	0.029	0.025	0.044	0.041	0.036	0.030	0.024	0.014	0.039	0.036	0.033
	SpinCalendar	0.023	0.020	0.014	0.045	0.038	0.030	0.212	0.201	0.178	0.283	0.273	0.263
1080p	BlueSky	0.286	0.241	0.035	0.400	0.376	0.360	0.507	0.476	0.446	1.008	0.904	0.673
	PedestrianArea	0.160	0.122	0.091	0.260	0.209	0.180	0.889	0.634	0.498	0.294	0.240	0.205
	RiverBed	0.050	0.042	0.025	0.218	0.168	0.147	1.291	1.062	0.919	0.076	0.053	0.031
	RushHour	0.080	0.060	0.046	0.324	0.273	0.231	2.282	1.818	1.496	0.176	0.141	0.112
Average		0.103	0.081	0.061	0.216	0.177	0.136	0.577	0.412	0.314	0.345	0.305	0.329

Table 6-1 pdf fitting to the predicted residue of each frame with different parameter estimation methods.

χ^2_{norm} for $C(0, 0)$		ML Laplace Est.			Cauchy Estimator 1			Proposed Cauchy Est.		
		Max.	Ave.	Min.	Max.	Ave.	Min.	Max.	Ave.	Min.
CIF	Foreman	0.138	0.120	0.100	0.052	0.033	0.022	0.054	0.036	0.028
	Football	0.055	0.032	0.022	0.031	0.018	0.010	0.225	0.143	0.072
	Paris	0.128	0.119	0.109	0.048	0.035	0.023	0.065	0.057	0.005
	Coastguard	0.022	0.016	0.011	0.066	0.047	0.030	0.641	0.528	0.405
	Mobile	0.091	0.081	0.069	0.078	0.053	0.037	0.141	0.126	0.111
WVGA	Flowervase	1.939	1.494	1.052	1.241	1.071	0.926	0.973	0.728	0.522
	Keiba	0.128	0.050	0.026	0.041	0.016	0.006	0.246	0.159	0.071
	Mobisode2	0.828	0.599	0.354	0.375	0.308	0.245	1.267	0.759	0.251
	RaceHorses	0.050	0.038	0.029	0.012	0.007	0.004	0.139	0.112	0.086
720p	City	0.187	0.172	0.162	0.054	0.048	0.041	0.057	0.052	0.047
	Crew	0.603	0.155	0.117	0.364	0.060	0.030	0.254	0.060	0.037
	Harbour	0.017	0.014	0.011	0.023	0.017	0.012	0.127	0.119	0.107
	SpinCalendar	0.289	0.273	0.253	0.058	0.053	0.047	0.056	0.051	0.045
1080p	BlueSky	1.062	0.997	0.939	0.524	0.466	0.417	0.714	0.655	0.599
	PedestrianArea	0.473	0.326	0.227	0.497	0.381	0.248	0.213	0.172	0.134
	RiverBed	0.035	0.015	0.002	0.057	0.040	0.025	0.997	0.643	0.377
	RushHour	0.077	0.069	0.054	0.069	0.045	0.026	0.342	0.250	0.207
Average		**0.360**	**0.269**	**0.208**	**0.211**	**0.159**	**0.126**	**0.383**	**0.274**	**0.183**

Table 6-2 pdf fitting to the transform coefficient $C(0, 0)$ in different sequences.

χ^2_{norm} for $C(1, 0)$		ML Laplace Est.			Cauchy Estimator 1			Proposed Cauchy Est.		
		Max.	Ave.	Min.	Max.	Ave.	Min.	Max.	Ave.	Min.
CIF	Foreman	0.203	0.172	0.141	0.068	0.050	0.035	0.040	0.031	0.023
	Football	0.033	0.024	0.015	0.079	0.039	0.026	0.197	0.127	0.077
	Paris	0.260	0.236	0.218	0.092	0.070	0.057	0.266	0.222	0.177
	Coastguard	0.026	0.020	0.016	0.090	0.064	0.041	1.992	1.750	1.455
	Mobile	0.325	0.262	0.211	0.253	0.200	0.158	0.164	0.133	0.109
WVGA	Flowervase	1.790	1.624	1.490	1.029	0.862	0.710	2.420	2.183	1.837
	Keiba	0.231	0.120	0.077	0.050	0.031	0.019	0.248	0.114	0.029
	Mobisode2	0.504	0.408	0.329	0.584	0.479	0.396	1.865	1.179	0.450
	RaceHorses	0.157	0.134	0.109	0.024	0.019	0.015	0.056	0.036	0.021
720p	City	0.266	0.245	0.213	0.186	0.135	0.105	0.115	0.088	0.061
	Crew	0.660	0.114	0.067	0.462	0.064	0.031	1.633	0.291	0.131
	Harbour	0.058	0.052	0.045	0.125	0.107	0.091	0.626	0.538	0.410
	SpinCalendar	0.674	0.648	0.616	0.119	0.112	0.101	0.177	0.163	0.154
1080p	BlueSky	1.341	1.149	0.918	0.255	0.174	0.092	1.045	0.871	0.667
	PedestrianArea	0.259	0.207	0.178	0.239	0.184	0.156	0.305	0.231	0.169
	RiverBed	0.042	0.026	0.011	0.135	0.105	0.042	2.186	1.742	1.449
	RushHour	0.103	0.077	0.050	0.088	0.066	0.042	0.169	0.106	0.067
Average		**0.408**	**0.325**	**0.277**	**0.228**	**0.162**	**0.125**	**0.794**	**0.577**	**0.429**

Table 6-3 pdf fitting to the transform coefficient $C(1, 0)$ in different sequences.

χ^2_{norm} for $C(0, 1)$		ML Laplace Est.			Cauchy Estimator 1			Proposed Cauchy Est.		
		Max.	Ave.	Min.	Max.	Ave.	Min.	Max.	Ave.	Min.
CIF	Foreman	0.173	0.148	0.117	0.065	0.054	0.036	0.038	0.030	0.020
	Football	0.138	0.061	0.034	0.052	0.032	0.019	0.096	0.064	0.020
	Paris	0.457	0.418	0.390	0.091	0.071	0.061	0.081	0.071	0.065
	Coastguard	0.031	0.022	0.016	0.067	0.046	0.034	0.139	0.105	0.066
	Mobile	0.220	0.183	0.154	0.185	0.147	0.105	0.105	0.087	0.070
WVGA	Flowervase	1.209	1.038	0.871	0.606	0.463	0.327	1.634	1.320	1.045
	Keiba	0.200	0.108	0.070	0.065	0.048	0.035	0.157	0.065	0.031
	Mobisode2	0.760	0.597	0.425	0.683	0.566	0.472	0.588	0.483	0.425
	RaceHorses	0.163	0.136	0.112	0.060	0.046	0.036	0.040	0.031	0.024
720p	City	0.371	0.354	0.326	0.220	0.191	0.168	0.168	0.122	0.100
	Crew	0.786	0.261	0.203	0.269	0.036	0.015	0.183	0.103	0.077
	Harbour	0.043	0.039	0.034	0.085	0.072	0.057	1.302	1.091	0.942
	SpinCalendar	0.401	0.382	0.362	0.105	0.093	0.083	0.154	0.141	0.129
1080p	BlueSky	1.130	1.001	0.902	0.125	0.075	0.052	0.724	0.533	0.347
	PedestrianArea	0.415	0.339	0.279	0.245	0.187	0.148	0.361	0.306	0.261
	RiverBed	0.104	0.058	0.014	0.186	0.141	0.044	0.621	0.376	0.186
	RushHour	0.160	0.125	0.096	0.061	0.037	0.012	0.789	0.575	0.373
Average		**0.398**	**0.310**	**0.259**	**0.186**	**0.136**	**0.100**	**0.422**	**0.324**	**0.246**

Table 6-4 pdf fitting to the transform coefficient $C(0, 1)$ in different sequences.

χ^2_{norm} for $C(0, 2)$		ML Laplace Est.			Cauchy Estimator 1			Proposed Cauchy Est.		
		Max.	Ave.	Min.	Max.	Ave.	Min.	Max.	Ave.	Min.
CIF	Foreman	0.210	0.168	0.125	0.098	0.070	0.050	0.571	0.434	0.303
	Football	0.351	0.155	0.074	0.064	0.037	0.014	0.665	0.140	0.013
	Paris	1.176	1.075	1.000	0.099	0.089	0.080	0.250	0.216	0.196
	Coastguard	0.034	0.028	0.022	0.064	0.053	0.042	0.186	0.129	0.077
	Mobile	0.314	0.275	0.225	0.156	0.132	0.105	0.117	0.095	0.076
WVGA	Flowervase	1.425	1.273	1.127	1.416	1.162	0.915	3.158	2.746	2.273
	Keiba	0.206	0.131	0.066	0.119	0.089	0.069	3.493	2.097	1.641
	Mobisode2	0.767	0.647	0.580	1.031	0.855	0.717	1.084	0.937	0.799
	RaceHorses	0.402	0.370	0.321	0.089	0.073	0.063	0.312	0.257	0.180
720p	City	0.350	0.326	0.302	0.180	0.160	0.147	0.441	0.210	0.107
	Crew	0.621	0.251	0.190	0.288	0.101	0.075	0.237	0.144	0.109
	Harbour	0.075	0.064	0.050	0.099	0.085	0.068	1.856	1.636	1.459
	SpinCalendar	0.247	0.229	0.212	0.119	0.105	0.092	0.304	0.276	0.253
1080p	BlueSky	0.395	0.327	0.234	0.236	0.133	0.149	1.599	1.308	0.984
	PedestrianArea	0.244	0.200	0.149	0.267	0.210	0.170	0.282	0.201	0.160
	RiverBed	0.130	0.094	0.073	0.117	0.091	0.071	0.464	0.295	0.169
	RushHour	0.290	0.188	0.110	0.191	0.130	0.091	0.304	0.231	0.201
Average		**0.426**	**0.341**	**0.286**	**0.273**	**0.210**	**0.172**	**0.901**	**0.668**	**0.529**

Table 6-5 pdf fitting to the transform coefficient $C(0, 2)$ in different sequences.

χ^2_{norm} for $C(1, 1)$		ML Laplace Est.			Cauchy Estimator 1			Proposed Cauchy Est.		
		Max.	Ave.	Min.	Max.	Ave.	Min.	Max.	Ave.	Min.
CIF	Foreman	0.187	0.148	0.129	0.039	0.030	0.018	0.064	0.043	0.030
	Football	0.101	0.049	0.023	0.050	0.030	0.013	0.201	0.120	0.045
	Paris	0.387	0.354	0.318	0.066	0.054	0.043	0.376	0.329	0.280
	Coastguard	0.018	0.013	0.009	0.048	0.041	0.033	0.854	0.664	0.502
	Mobile	0.353	0.311	0.276	0.267	0.216	0.171	0.173	0.148	0.130
WVGA	Flowervase	1.247	1.117	0.995	1.374	1.131	0.887	2.580	2.260	1.945
	Keiba	0.183	0.120	0.085	0.047	0.036	0.031	0.088	0.043	0.028
	Mobisode2	0.539	0.459	0.398	0.707	0.591	0.493	3.266	2.298	1.401
	RaceHorses	0.252	0.229	0.203	0.049	0.038	0.031	0.049	0.041	0.030
720p	City	0.218	0.199	0.169	0.177	0.149	0.125	0.136	0.111	0.096
	Crew	0.416	0.085	0.055	0.383	0.075	0.047	0.327	0.136	0.061
	Harbour	0.044	0.037	0.025	0.084	0.072	0.045	0.221	0.141	0.111
	SpinCalendar	0.274	0.261	0.246	0.089	0.080	0.074	0.089	0.082	0.073
1080p	BlueSky	0.581	0.469	0.344	0.268	0.177	0.091	0.369	0.250	0.120
	PedestrianArea	0.192	0.169	0.142	0.250	0.222	0.196	0.605	0.476	0.395
	RiverBed	0.056	0.040	0.024	0.100	0.083	0.050	1.558	1.194	0.920
	RushHour	0.126	0.099	0.080	0.159	0.128	0.106	1.870	1.702	1.584
Average		**0.304**	**0.245**	**0.207**	**0.245**	**0.185**	**0.144**	**0.754**	**0.590**	**0.456**

Table 6-6 pdf fitting to the transform coefficient $C(1, 1)$ in different sequences.

χ^2_{norm} for $C(2, 0)$		ML Laplace Est.			Cauchy Estimator 1			Proposed Cauchy Est.		
		Max.	Ave.	Min.	Max.	Ave.	Min.	Max.	Ave.	Min.
CIF	Foreman	0.212	0.174	0.148	0.063	0.044	0.028	0.126	0.083	0.052
	Football	0.057	0.036	0.018	0.086	0.063	0.039	0.101	0.057	0.032
	Paris	0.473	0.405	0.373	0.102	0.091	0.075	0.681	0.566	0.468
	Coastguard	0.019	0.013	0.009	0.106	0.054	0.036	3.941	3.979	2.862
	Mobile	0.389	0.344	0.308	0.278	0.212	0.170	0.194	0.171	0.152
WVGA	Flowervase	1.981	1.808	1.582	1.606	1.346	1.110	2.113	1.915	1.623
	Keiba	0.361	0.236	0.173	0.052	0.036	0.026	0.283	0.129	0.046
	Mobisode2	0.614	0.531	0.461	0.796	0.673	0.561	1.286	0.807	0.456
	RaceHorses	0.286	0.260	0.229	0.051	0.042	0.036	0.415	0.341	0.233
720p	City	0.427	0.386	0.323	0.223	0.165	0.134	0.275	0.172	0.093
	Crew	0.502	0.091	0.057	0.437	0.083	0.052	1.186	0.163	0.060
	Harbour	0.111	0.102	0.078	0.129	0.110	0.076	2.419	1.969	1.275
	SpinCalendar	0.520	0.489	0.449	0.089	0.078	0.070	0.092	0.080	0.067
1080p	BlueSky	0.774	0.631	0.462	0.245	0.178	0.100	0.707	0.578	0.483
	PedestrianArea	0.208	0.179	0.159	0.250	0.216	0.183	0.236	0.189	0.162
	RiverBed	0.035	0.027	0.017	0.044	0.039	0.026	2.035	1.857	1.724
	RushHour	0.115	0.100	0.091	0.147	0.132	0.121	0.438	0.296	0.194
Average		**0.417**	**0.342**	**0.290**	**0.277**	**0.210**	**0.167**	**0.972**	**0.785**	**0.587**

Table 6-7 pdf fitting to the transform coefficient $C(2, 0)$ in different sequences.

6.8 References

[1] Randall C. Reininger and Jerry D. Gibson, "*Distributions of Two-Dimensional DCT Coefficients for Images,*" IEEE Trans. on Communications, vol. COM-31, no. 6, pp. 835-839, June 1983.

[2] F. Muller, "*Distribution shape of Two-dimensional DCT coefficients of natural images,*" IEEE Electronic Letters, vol. 29, issue 22, pp. 1935-1936, 1993.

[3] T. Eude, R. Grisel, H. Cherifi and R. Debrie, "*On the Distribution of the DCT coefficients,*" IEEE ICASSP-94, vol. 5, pp. 365-368, 1994.

[4] E. Y. Lam and J. W. Goodman, "*A Mathematical Analysis of the DCT Coefficient Distributions for Images,*" IEEE Trans. on IP, vol. 9, issue 10, pp. 1661-1666, Oct. 2000.

[5] E. Y. Lam, "*Analysis of the DCT Coefficient Distributions for Document Coding,*" IEEE Signal Processing Letters, vol. 11, no. 2, pp. 97-100, Feb. 2004.

[6] Joon-Hyuk Chang, Jon Won Shin, Nam Soo Kim and Sanjit K. Mitra, "*Image Probability Distribution Based on Generalized Gamma Function,*" IEEE Signal Processing Letters, vol. 12, no. 4, pp. 325-328, Apr. 2005.

[7] Saralees Nadarajah and Samuel Kotz, "*On the DCT Coefficient Distributions,*" IEEE Signal Processing Letters, vol. 13, issue 10, pp. 601-603, Oct. 2006.

[8] Y. Altunbasak and N. Kamaci, "*An analysis of the DCT coefficient distribution with the H.264 video coder,*" IEEE ICASSP -04, vol. 3, pp. 177-180, 2004.

[9] N. Kamaci, Y. Altunbasak and Russell M. Mersereau, "*Frame Bit Allocation for the H.264/AVC Video Coder via Cauchy-Density-Based Rate and Distortion Models,*" IEEE Trans. on CASVT, vol. 15, no. 8 pp. 994-1006, Aug. 2005.

[10] K. Krishnamoorthy, Handbook of Statistical Distributions with Applications, CRC Press. 2006.

Chapter 7 Summary and Future Work

7.1 Contributions

7.2.1 Order-16 DCT-like Transforms

In this thesis, we have developed 3 classes of orthogonal order-16 DCT-like integer transforms. They are Simple Integer Transform, Hybrid Integer Transform and Integer Transform from Relaxed General Cosine Transform (RGCT). Simple Integer Transform is extended from order-8 Integer Cosine Transform (ICT). It has a very simple structure and it requires very little computation. It also has a good coding performance. In order to improve the coding performance, Hybrid Integer Transform is proposed. It is a hybrid of ICT and Dyadic Weight Walsh Transform (DWWT). It has a better coding performance than Simple Integer Transform. We proposed a method to derive ICT from RGCT. All are orthogonal, similar to the DCT and have fast algorithms. In this class of integer transform, LLMICT and CSFICT are proposed. LLMICT is an ICT having a fast algorithm similar to the LLM Fast DCT algorithm which is proposed by Loeffler et al. It has an excellent

coding performance. However, it does not require heavy computation. It is also possible to extend it to higher order transform, such as order-32. CSFICT has an algorithm similar to the CSF Fast DCT algorithm proposed Chen et al. We modify it to become MCSFICT which has a looser criterion for orthogonality. This leads to a high flexibility of designing high performance order-16 ICT. Experiments show that it has a performance similar to LLMICT.

These proposed order-16 transforms are integrated into the reference software of two video coding standards, H.264/AVC and AVS. Together with order-8 and order-4 transform in these standards, two different ABT platforms are formed. They are tested and compared with other existing order-16 transforms. Experiments show that these proposed transforms provide a significant gain, especially for HD sequences, in the two standards. Not only the objective coding performance, but also the subjective quality is improved. The proposed transforms perform similar to other order-16 transforms but they are significantly simpler.

7.2.2 Fast Walsh Search for Pattern Matching

Mak and Li proposed a fast pattern matching algorithm in Walsh Hadamard domain. It has a high speed and high accuracy. It was integrated into H.264/AVC to do the motion estimation. To speed up the matching process, we proposed a statistical threshold which can be adjusted according to the desired accuracy. This threshold dramatically reduces the number of candidates which are possibly mismatched. As a result, the matching process is significantly speeded up without missing the best match.

Picture nature varies from region to region. Some are smooth while some are textured. We expected that the proposed statistical threshold should be adaptive to the nature of the image content. As a result, a block adaptive threshold was proposed to eliminate the mismatch candidates more efficiently with the same accuracy. This threshold depends on the variance of the target block. Experiment shows that the mismatch candidates are reduced. Unfortunately, the time saving is not significant as computing block variance is necessary.

7.2.3 Transform Coefficient Distribution

A preliminary study of the transform coefficient distribution of the predicted residue was carried out. We found that the intra- and the inter-predicted residues have different distributions. Intra one is closer to Cauchy distribution while inter one is closer to Laplace distribution. As a result their transform coefficients are closer to Cauchy distribution and Laplace distribution respectively.

Methods to estimate the distribution parameters of the transform coefficient are proposed. They are compared with experiments. It is shown that the accuracies of these methods are pretty high.

7.1 Future Work

7.1.1 Order-16 DCT-like Transforms

There are a lot of work can be done on the proposed DCT-like transforms. For example, we have proposed order-32 LLMICT but have not tested it yet. It can be integrated into our proposed platform to achieve a more powerful ABT platform for

HD video coding. In our proposed ABT platform, it is noticed that there is a high correlation between the DC coefficients of neighbouring MB. It is possible to improve the performance by removing this correlation. Using the development idea of deriving ICT from RGCT, it is possible to develop other trigonometric integer transforms of higher orders with fast algorithms. This will be very useful in many image and video analysis applications.

7.1.2 Fast Walsh Search for Pattern Matching

The proposed block adaptive threshold significantly reduces the mismatch candidates in pattern matching. However, the time saving is limited by the block variance calculation at this moment. It is worth to find methods to speed up this calculation so that a very high speed pattern matching algorithm can be developed.

Proposed FWS has been implemented into H.264/AVC as a motion estimation tools. Currently, the candidates are found only by minimizing the difference (in terms of MSE or MAE) between the reference block and the target block. The resultant motion vector (MV) is not a factor of selection. It is possible to improve the performance by considering the resultant MV in a RD-optimized manner. MV are usually predicted by a median predictor using MV of neighbouring MB. A candidate's MV which is more different from the predicted MV requires more bits to code it. This candidate may not be a RD-optimized candidate even it have similarly small MSE or MAE as other candidates. Thus, it can be eliminated from the candidate pool and hence the pool size is reduced.

7.1.3 Transform Coefficient Distribution

The transform coefficient distribution analysis is in a very beginning stage. We have only verified our findings in many test sequences in different resolutions and nature. However, there is no application developed based on our findings. Fortunately, many existing transform based image and video processing applications are developed under the assumption that the coefficient distribution is Laplace distribution, for example, the RD optimization in video coding. An improvement is expected when they are redesigned with our findings.

Appendix A. Fast Algorithm for DWWT

A.1 Factorization of Matrix with Orthogonal Basis Vectors

For any $n \times n$ matrix $\mathbf{E}$ with orthogonal basis vectors which can be decomposed into two $n \times n$ matrices $\mathbf{A}$ and $\mathbf{B}$. Assume that $\mathbf{A}$ and $\mathbf{B}$ also have orthogonal basis vectors. Therefore we have:

$$\mathbf{E} = \mathbf{AB}, \tag{A-1}$$

$$\mathbf{EE}^T = \mathbf{D}_E, \tag{A-2}$$

$$\mathbf{AA}^T = \mathbf{D}_A, \tag{A-3}$$

$$\mathbf{BB}^T = \mathbf{D}_B. \tag{A-4}$$

where $\mathbf{D}_E$, $\mathbf{D}_A$ and $\mathbf{D}_B$ are diagonal matrices. From (A-1) and (A-2), it can be expended to:

$$\begin{aligned} \mathbf{EE}^T = \mathbf{D}_E &= \mathbf{ABB}^T\mathbf{A}^T \\ &= \mathbf{AD}_B\mathbf{A}^T. \end{aligned} \tag{A-5}$$

By multiplying $\mathbf{A}^T$ and $\mathbf{A}$, we have:

$$\mathbf{A}^T\mathbf{D}_E\mathbf{A} = \mathbf{D}_A\mathbf{D}_B\mathbf{D}_A = \mathbf{D}_A^2\mathbf{D}_B. \tag{A-6}$$

we can rewrite it as:

$$\mathbf{A}^T\mathbf{D}_E^{1/2}\mathbf{D}_E^{1/2}\mathbf{A} = \mathbf{D}_A\mathbf{D}_B\mathbf{D}_A = \mathbf{D}_A^2\mathbf{D}_B . \tag{A-7}$$

and $\mathbf{D}_E^{1/2}\mathbf{A}$ is an orthogonal matrix also and the left hand side of (A-7) is equal to $\mathbf{D}_A\mathbf{D}_E$. Therefore:

$$\mathbf{D}_E = \mathbf{D}_A\mathbf{D}_B . \tag{A-8}$$

i.e.

$$d_E(i) = d_A(i)d_B(i) . \tag{A-9}$$

This means if (A-1) to (A-4) are fulfilled, the norm of the i-th row vector of $\mathbf{E}$ defined as:

$$\left\|\vec{E}(i)\right\| = \vec{E}(i)\cdot\vec{E}(i)^T = d_E(i) \tag{A-10}$$

can be represented by a product of the norms of the i-th row vectors of $\mathbf{A}$ and $\mathbf{B}$ as shown in (A-9). Recusively, if $\mathbf{E}$ can be factorized into product of k matrices,

$$\mathbf{T} = \mathbf{M}_0\mathbf{M}_1...\mathbf{M}_k \tag{A-11}$$

from (A-8) and (A-9), its norms can be expressed as:

$$\mathbf{D}_T = \mathbf{D}_{M_0}\mathbf{D}_{M_1}...\mathbf{D}_{M_k} , \tag{A-12}$$

$$\text{or} \quad d_T(i) = d_{M_0}(i)d_{M_1}(i)...d_{M_k}(i) . \tag{A-13}$$

For more specific, if $\mathbf{E}$ is an integer kernel of a DWWT which is going to be factorized into integer matrices $\mathbf{M}_p \in \{\mathbf{M}_0, \mathbf{M}_1, ..., \mathbf{M}_k\}$, these $\mathbf{M}_p$ can be found in a

finite search space. If the norm of **E** can be factorized into several prime factors, these prime factors (or the products of some of these prime factors) are very likely to be the norms of the factors $\mathbf{M}_p$. As a result, this significantly reduces the search space for finding $\mathbf{M}_p$. It is not necessary that:

$$d_{Mp}(i) = d_{Mp}(j) \quad \text{for any} \quad 0 \le p < k \quad \text{and} \quad 0 \le i, j < n \tag{A-14}$$

or

$$\left|\det(\mathbf{M}_p)\right| = \left[d_{Mp}(i)\right]^8. \tag{A-15}$$

which is specified in [1]. (A-15) holds only when $n = 8$ and $d_{Mp}(i)$ is constant for all $0 \le i < n$ which means all the row vectors have the same norm. If it is assumed that (A-14) holds, this will turn into a generalized version in [1] with any order $n > 1$.

Consider a case that the norm of **E**, $d_E(i)$, is a prime number. **E** cannot be factorized into product of integer matrices with the above method. If **E** is a DWWT, **E** can be broken down into simpler matrices to achieve fast matrix multiplication. Let us take order-8 DWWT as an example. If $\mathbf{E} = \mathbf{E}_{DWW}(b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7)$ where $d_E = \sum b_j^2$ is a prime number, the decomposition can be achieved by replace b_j by $(b_j + k)$ for certain j. And k is a small non-zero integer such as 1 or -1. This replacement turns **E** into $\mathbf{E}' = \mathbf{E}_{DWW}(b_0, \ldots, b_j + k, \ldots, b_7)$ which $d_{E'}$ is a compound number instead of a prime one. $\mathbf{E}_{DWW}(5, 6, 5, 4, 4, 3, 3, 1)$ has a norm of 137 which is a prime number. It cannot be decomposed directly. However, $\mathbf{E}_{DWW}(5, 5, 5, 4, 4, 3, 3, 1)$ can be decomposed into 4 sparse matrices which requires 8 shift and 64 addition operations:

$\mathbf{E}_{DWW}(5,5,5,4,4,3,3,1)$

$$= \begin{bmatrix} 2 & & 1 & & & & -1 & 1 \\ 1 & & & -1 & & -1 & & -2 \\ & 2 & 1 & & 1 & & 1 & \\ & & & & -1 & -2 & 1 & 1 \\ 1 & -1 & & & & 1 & 2 & \\ & & 1 & 2 & -1 & & & -1 \\ 1 & 1 & -2 & 1 & & & & \\ & 1 & & -1 & -2 & 1 & & \end{bmatrix} \begin{bmatrix} 1 & & 1 & & 1 & & & \\ 1 & & & & & -1 & -1 & \\ 1 & & -1 & & & & 1 & \\ & & & 1 & -1 & & 1 & \\ & & 1 & & & 1 & & \\ & & & & 1 & & & 1 \\ & & 1 & & -1 & & & \\ & & & & 1 & & 1 & 1 \end{bmatrix}$$

$$\times \begin{bmatrix} 1 & 1 & & & 1 & & & \\ 1 & & -1 & & -1 & & & \\ 1 & -1 & 1 & & & & & \\ & 1 & 1 & & -1 & & & \\ & & & 1 & & 1 & 1 & \\ & & & 1 & & & -1 & 1 \\ & & & 1 & & -1 & & -1 \\ & & & & & 1 & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & & & & & & & 1 \\ & & & 1 & & & 1 & \\ & 1 & & & & & & \\ & & 1 & & & 1 & & \\ & & & & 1 & & & \\ 1 & & & & & & & \\ & & & 1 & & & -1 & \\ & & & & 1 & -1 & & \end{bmatrix} \tag{A-16}$$

$$\begin{aligned} &\mathbf{E}_{DWW}(5,6,5,4,4,3,3,1) \\ &= \mathbf{E}_{DWW}(5,5,5,4,4,3,3,1) + \mathbf{E}_{DWW}(0,1,0,0,0,0,0,0) \end{aligned} \tag{A-17}$$

Since DWWT is linear, $\mathbf{E}_{DWW}(5, 6, 5, 4, 4, 3, 3, 1)$ can be expressed into two matrices as shown (A-17). As a result, $\mathbf{E}_{DWW}(5, 6, 5, 4, 4, 3, 3, 1)$ can be expressed into sparse matrices which can be computed with 8 shift and 72 addition operations indirectly.

This technique is not only useful to those **E** having a prime number $d_E(i)$, but also to those **E** which $d_E(i)$ cannot be factorized into small prime factors. $\mathbf{E}_{DWW}(5, 5, 5, 4, 4, 3, 2, 1)$ has a norm of $121 = 11 \times 11$. It can only be decomposed into two matrices which are not very sparse as shown in (A-18). Its fast algorithm requires 16 shift and 77 addition operations.

$\mathbf{E}_{DWW}(5,5,5,4,4,3,2,1)$

$$= \begin{bmatrix} 1 & 1 & -1 & 1 & 1 & 1 & -1 & -2 \\ & 1 & & 2 & -2 & 1 & & 1 \\ & -2 & & 2 & & -1 & 1 & -1 \\ & & 2 & 1 & 2 & 1 & & 1 \\ 1 & -1 & 1 & -1 & -1 & 2 & 1 & -1 \\ 3 & & & & & -1 & & 1 \\ & -2 & -1 & & & 1 & -2 & 1 \\ & & -2 & & 1 & 1 & 2 & 1 \end{bmatrix} \begin{bmatrix} 2 & -1 & -1 & 2 & & 1 & & \\ -1 & 1 & 1 & 2 & & & & -2 \\ -1 & 1 & -2 & & & 1 & -2 & \\ 2 & 2 & & -1 & -1 & & & -1 \\ & & & -1 & 2 & 2 & 1 & -1 \\ 1 & & & & 2 & -1 & -2 & \\ & & -2 & & 1 & -2 & 1 & -1 \\ -2 & & & -1 & -1 & & -1 & -2 \end{bmatrix} \tag{A-18}$$

However, its "neighbor", $\mathbf{E}_{DWW}(5, 5, 5, 4, 4, 3, 2+1, 1)$ has a norm of $126 = 2 \times 3 \times 3 \times 7$. It can be decomposed into 4 sparse matrices as shown in (A-16). Plus 8 addition operations to restore $\mathbf{E}_{DWW}(5, 5, 5, 4, 4, 3, 2, 1)$, this indirect method requires only 8 shift and 72 addition operations. It saves 8 shift and 5 addition operations.

$$\begin{aligned}&\mathbf{E}_{DWW}(5,5,5,4,4,3,2,1)\\&= \mathbf{E}_{DWW}(5,5,5,4,4,3,3,1) - \mathbf{E}_{DWW}(0,0,0,0,0,0,1,0)\end{aligned} \quad \textbf{(A-19)}$$

$\mathbf{E}_{DWW}$ becomes commoner to have a prime $d_7(i)$ or $d_7(i)$ which cannot be factorized into simple factors when b_i are getting large. Therefore, this indirect method is a very useful and efficient way to decompose $\mathbf{E}_{DWW}$ with large and prime b_i.

A.2 Reference

[1] Jie Dong; King Ngi Ngan; Chi-Keung Fong; Wai-Kuen Cham, "*2-D Order-16 Integer Transforms for HD Video Coding*," IEEE Trans. on CASVT, vol. 19, Issue: 10, pp. 1462 – 1474, Oct. 2009.

Appendix B. A Summary of Fast Algorithms for Different Integer Transforms

In this appendix, the fast algorithms of different integer transforms are described. Their matrix factorization and their number of operations will be stated. The fast algorithms described in this appendix include:

- Order-8 ICT adopted in H.264/AVC,
- Order-8 ICT adopted in AVS,
- Modified ICT (MICT) [4],
- Integer Transform proposed by Wien et al [1], $\mathbf{T}_{Wien}$,
- Integer Transform proposed by Lee and et al [2], $\mathbf{T}_{Lee}$,
- Integer Transform proposed by Joshi and et al [3], $\mathbf{T}_{Joshi}$,
- Proposed Order-16 Simple Integer Transform, $\mathbf{T}_{SI\text{-}AVS}$ and $\mathbf{T}_{SI\text{-}H264}$,
- Proposed Order-16 Hybrid Integer Transforms, $\mathbf{T}_{HI1}$ and $\mathbf{T}_{HI2}$, and
- Proposed LLMICT, A1 and B1.

Here we declare some common notations in this appendix:

$\mathbf{0}_N$	Order-N zero matrix.
$\mathbf{I}_N$	Order-N identity matrix.
$\tilde{\mathbf{I}}_N$	$\mathbf{I}_N$ rotated by 90°.
$\mathbf{H}_N$	$N\times N$ order-2 Hadamard matrix. $\begin{bmatrix} \mathbf{I}_{N/2} & \tilde{\mathbf{I}}_{N/2} \\ \mathbf{I}_{N/2} & -\tilde{\mathbf{I}}_{N/2} \end{bmatrix}$

B.1 Fast Order-8 ICT in H.264/AVC

This implementation can be easily found in the reference software of H.264/AVC. $ICT_8(8, 12, 10, 6, 3, 8, 4)$ is adopted:

$$\mathbf{E}_{H264} = \begin{bmatrix} 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 \\ 12 & 10 & 6 & 3 & -3 & -6 & -10 & -12 \\ 8 & 4 & -4 & -8 & -8 & -4 & 4 & 8 \\ 10 & -3 & -12 & -6 & 6 & 12 & 3 & -10 \\ 8 & -8 & -8 & 8 & 8 & -8 & -8 & 8 \\ 6 & -12 & 3 & 10 & -10 & 3 & 12 & -6 \\ 4 & -8 & 8 & -4 & -4 & 8 & -8 & 4 \\ 3 & -6 & 10 & -12 & 12 & -10 & 6 & -3 \end{bmatrix} \tag{B.1}$$

It can be decomposed into:

$$\mathbf{E}_{H264} = \begin{bmatrix} \begin{bmatrix} 8 & 8 \\ 8 & -8 \end{bmatrix} & \mathbf{0}_2 & & \mathbf{0}_4 \\ \mathbf{0}_2 & \begin{bmatrix} 4 & 8 \\ -8 & 4 \end{bmatrix} & & \\ & & \begin{bmatrix} 4 & 0 & 0 & 1 \\ 0 & -1 & -4 & 0 \\ 0 & -4 & 1 & 0 \\ 1 & 0 & 0 & -4 \end{bmatrix} & \\ \mathbf{0}_4 & & & \end{bmatrix} \begin{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & -1 & 0 \\ 1 & 0 & 0 & -1 \end{bmatrix} & & & \mathbf{0}_4 \\ & & & \\ & & & \\ \mathbf{0}_4 & & & \begin{bmatrix} 3 & 2 & 2 & 0 \\ -2 & 3 & 0 & -2 \\ -2 & 0 & 3 & 2 \\ 0 & 2 & -2 & 3 \end{bmatrix} \end{bmatrix} \begin{bmatrix} \mathbf{I}_4 & \tilde{\mathbf{I}}_4 \\ \mathbf{I}_4 & -\tilde{\mathbf{I}}_4 \end{bmatrix} \tag{B.2}$$

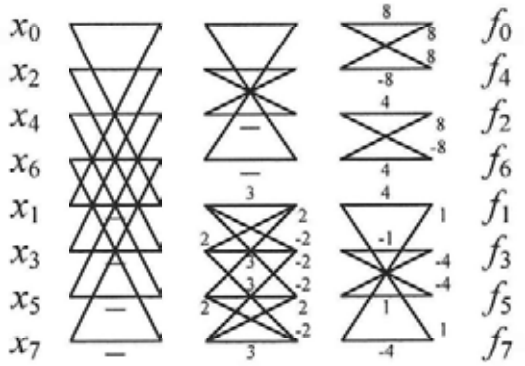


Figure B-1 Data flow of Order-8 ICT adopted in H.264/AVC

The fast algorithm requires 14 multiplications and 32 additions. In multiplication-free implementation, it requires 14 shifts and 32 additions (total 46 operations).

B.2 Fast Order-8 ICT in AVS

This implementation can be easily found in the reference software of AVS. $ICT_8(8, 10, 9, 6, 2, 10, 4)$ is adopted:

$$\mathbf{E}_{AVS} = \begin{bmatrix} 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 \\ 10 & 9 & 6 & 2 & -2 & -6 & -9 & -10 \\ 10 & 4 & -4 & -10 & -10 & -4 & 4 & 10 \\ 9 & -2 & -10 & -6 & 6 & 10 & 2 & -9 \\ 8 & -8 & -8 & 8 & 8 & -8 & -8 & 8 \\ 6 & -10 & 2 & 9 & -9 & 2 & 10 & -6 \\ 4 & -10 & 10 & -4 & -4 & 10 & -10 & 4 \\ 2 & -6 & 9 & -10 & 10 & -9 & 6 & -2 \end{bmatrix} \tag{B.3}$$

It can be decomposed into:

$$\mathbf{E}_{AVS} = \begin{bmatrix} 8 & 8 & \mathbf{0}_2 & & \mathbf{0}_4 & \\ 8 & -8 & & & & \\ \mathbf{0}_2 & & 4 & 10 & & \\ & & -10 & 4 & & \\ & \mathbf{0}_4 & & & 2 & 3 & 0 & 2 \\ & & & & 3 & -2 & -2 & 0 \\ & & & & 0 & -2 & 2 & 3 \\ & & & & 2 & 0 & 3 & -2 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 1 & \mathbf{0}_4 & \\ 0 & 1 & 1 & 0 & & \\ 0 & 1 & -1 & 0 & & \\ 1 & 0 & 0 & -1 & & \\ & \mathbf{0}_4 & & & 3 & 0 & 0 & -2 \\ & & & & 0 & 3 & 2 & 0 \\ & & & & 0 & -2 & 3 & 0 \\ & & & & 2 & 0 & 0 & 3 \end{bmatrix} \begin{bmatrix} \mathbf{I}_4 & \tilde{\mathbf{I}}_4 \\ \mathbf{I}_4 & -\tilde{\mathbf{I}}_4 \end{bmatrix} \tag{B.4}$$

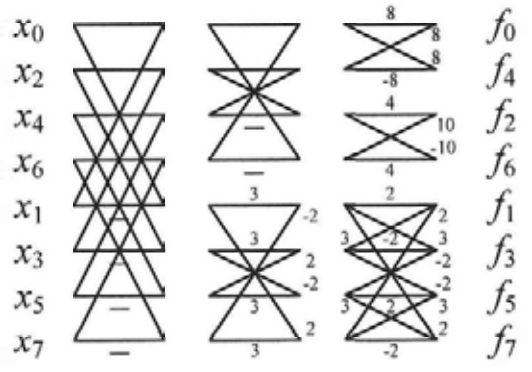


Figure B-2 Data flow of Order-8 ICT adopted in AVS.

The fast algorithm requires 14 multiplications and 36 additions. In multiplication-free implementation, it requires 16 shifts and 38 additions (total 54 operations).

B.3 Modified ICT

The integer kernel of modified ICT, $\mathbf{E}_{MICT}$, can be factorized into

$$\mathbf{E}_{MICT} = \begin{bmatrix} \mathbf{E}_{AVS} & \mathbf{0}_8 \\ \mathbf{0}_8 & \mathbf{P}_{MICT} \end{bmatrix} \begin{bmatrix} \mathbf{I}_8 & \tilde{\mathbf{I}}_8 \\ \mathbf{I}_8 & -\tilde{\mathbf{I}}_8 \end{bmatrix} \tag{B.5}$$

where

$$\mathbf{P}_{MICT} = \begin{bmatrix} -2 & & 1 & -1 & -1 & 3 & -1 & \\ 3 & -1 & 1 & 1 & & 2 & & 1 \\ -1 & -3 & 1 & & 1 & & 2 & -1 \\ & 1 & & 1 & 3 & 1 & -1 & -2 \\ 1 & -1 & -3 & -2 & & 1 & & -1 \\ 1 & 1 & 1 & & -2 & & 1 & -3 \\ & -2 & & 1 & -1 & -1 & -3 & -1 \\ -1 & & -2 & 3 & -1 & 1 & 1 & \end{bmatrix} \begin{bmatrix} & & & & 1 & -1 & & 1 \\ & & & 1 & 1 & & & \\ & & & & & & 1 & \\ & -1 & & -1 & & & 1 & \\ & -1 & 1 & & & & -1 & \\ & -1 & -1 & 1 & & & & \\ & & & & -1 & & -1 & \\ & & & & & -1 & & 1 \\ & & & & 1 & 1 & & \end{bmatrix} \begin{bmatrix} & -2 & & -1 & & 2 & -1 & 1 \\ & -1 & & 2 & 1 & -1 & & 2 \\ & -2 & 1 & 1 & & & 1 & -2 \\ -2 & & -1 & & 2 & & -1 & -1 \\ -1 & & 2 & & -1 & -1 & -2 & \\ -2 & & 1 & -1 & & & 2 & 1 \\ -1 & 1 & & 2 & -1 & 2 & & \\ -1 & -1 & -2 & & -2 & -1 & & \end{bmatrix} \tag{B.6}$$

The fast algorithm requires 30 multiplications and 144 additions. In multiplication-free implementation, it requires 30 shifts and 144 additions (total 174 operations).

B.4 Integer Transform proposed by Wien and Sun.

No fast algorithm has been proposed for $\mathbf{T}_{Wien}$. Here, we proposed its fast algorithm as below. The integer kernel for $\mathbf{T}_{Wien}$, $\mathbf{E}_{Wien}$, can be factorized into:

$$\mathbf{E}_{Wien} = \begin{bmatrix} \mathbf{Q}_{Wien} & \mathbf{0}_8 \\ \mathbf{0}_8 & \mathbf{P}_{Wien} \end{bmatrix} \mathbf{H}_{16}, \tag{B-7}$$

$$\mathbf{Q}_{Wien} = \begin{bmatrix} 17\mathbf{H}_2 & \mathbf{0}_2 & & \\ \mathbf{0}_2 & \begin{matrix} 7 & 23 \\ -23 & 7 \end{matrix} & \mathbf{0}_4 & \\ & & \begin{matrix} 4 & & & 1 \\ & -1 & -4 & \\ & -4 & 1 & \\ 1 & & & -4 \end{matrix} & \\ \mathbf{0}_4 & & & \end{bmatrix} \begin{bmatrix} \mathbf{H}_4 & \mathbf{0}_4 & & \\ & \begin{matrix} 3 & 2 & 2 & \\ -2 & 3 & & -2 \\ -2 & & 3 & 2 \\ & 2 & -2 & 3 \end{matrix} & & \\ \mathbf{0}_4 & & \begin{matrix} \mathbf{I}_4 & \widetilde{\mathbf{I}}_4 \\ 2\mathbf{I}_4 & -2\widetilde{\mathbf{I}}_4 \end{matrix} & \end{bmatrix}, \tag{B-8}$$

$$\mathbf{P}_{Wien} = 2\times \begin{bmatrix} -1 & 3 & -1 & 2 & & -1 & 1 & \\ 1 & & & 1 & 1 & -1 & -2 & -3 \\ 1 & -1 & & 3 & 1 & 2 & & 1 \\ & -1 & 1 & 1 & & -3 & -1 & 2 \\ -2 & 1 & 3 & & 1 & 1 & -1 & \\ 1 & & 2 & 1 & -3 & & 1 & -1 \\ 3 & 2 & 1 & -1 & 1 & & & 1 \\ & 1 & -1 & & -2 & 1 & -3 & 1 \end{bmatrix} \begin{bmatrix} 1 & -2 & 2 & -2 & 2 & & & \\ 2 & 1 & 2 & 2 & & -2 & & \\ 2 & -2 & -1 & & -2 & & 2 & \\ 2 & 2 & -2 & -1 & & 2 & & -2 \\ 2 & & & & 1 & & -2 & 2 \\ & 2 & 2 & -2 & & -1 & 2 & 2 \\ & & & & -2 & 2 & -1 & 2 \\ & & & 2 & 2 & 2 & 2 & 1 \end{bmatrix} \mathbf{H}_8, \tag{B-9}$$

The fast algorithm requires 34 multiplications and 136 additions. In multiplication-free implementation, it requires 38 shifts and 144 additions (total 182 operations).

B.5 Integer Transform proposed by Lee and et al.

No fast algorithm has been proposed for $\mathbf{T}_{Lee}$. Here, we proposed its fast algorithm as below. Its integer kernel $\mathbf{E}_{Lee}$ can be factorized into:

$$\mathbf{E}_{Lee} = \begin{bmatrix} \mathbf{Q}_{Lee} & \mathbf{0}_8 \\ \mathbf{0}_8 & \mathbf{P}_{Lee} \end{bmatrix} \mathbf{H}_{16}. \tag{B-10}$$

$$\mathbf{Q}_{Lee} = \left[\begin{array}{cc|cc|cc} 16\mathbf{H}_2 & \mathbf{0}_2 & & & & \\ & 7\ \ 17 & & & & \\ \mathbf{0}_2 & -17\ \ 7 & & \mathbf{0}_4 & & \\ \hline & & 5 & 7 & -5 & \\ & & & -5 & -5 & -7 \\ & \mathbf{0}_4 & 7 & -5 & 5 & \\ & & -5 & & 7 & -5 \end{array}\right] \left[\begin{array}{cc|cc} \mathbf{H}_4 & & \mathbf{0}_4 & \\ & & 3 & 2 \\ \hline & & 2\ \ 3 & \\ \mathbf{0}_4 & & 3\ -2 & \\ & & 2 & -3 \end{array}\right] \mathbf{H}_8 \tag{B-11}$$

$$\mathbf{P}_{Lee} = \begin{bmatrix} -9 & -7 & 9 & 9 & -3 & & 7 & 3 \\ 7 & 3 & -3 & & -9 & -9 & 9 & 7 \\ 9 & 3 & 9 & 7 & -7 & 3 & -9 & \\ -9 & & -7 & 3 & -9 & -7 & -9 & -3 \\ -3 & 9 & -7 & 9 & 3 & 7 & & 9 \\ & -9 & -3 & -7 & -7 & 9 & -3 & 9 \\ 7 & -9 & -9 & 9 & & 3 & 3 & -7 \\ -3 & 7 & & -3 & -9 & 9 & 7 & -9 \end{bmatrix} \begin{bmatrix} 1 & & & -2 & -2 & & & \\ & 1 & -2 & & & -2 & & \\ & 2 & 1 & & & & 2 & \\ 2 & & & 1 & & & & 2 \\ 2 & & & & 1 & & & -2 \\ & 2 & & & & 1 & -2 & \\ & & -2 & & & 2 & 1 & \\ & & & -2 & 2 & & & 1 \end{bmatrix} \tag{B-12}$$

The fast algorithm requires 50 multiplications and 112 additions. In multiplication-free implementation, it requires 58 shifts and 148 additions (total 206 operations).

B.6 Integer Transform proposed by Joshi and et al.

The fast algorithm proposed in [3] is as shown in Figure B-3.

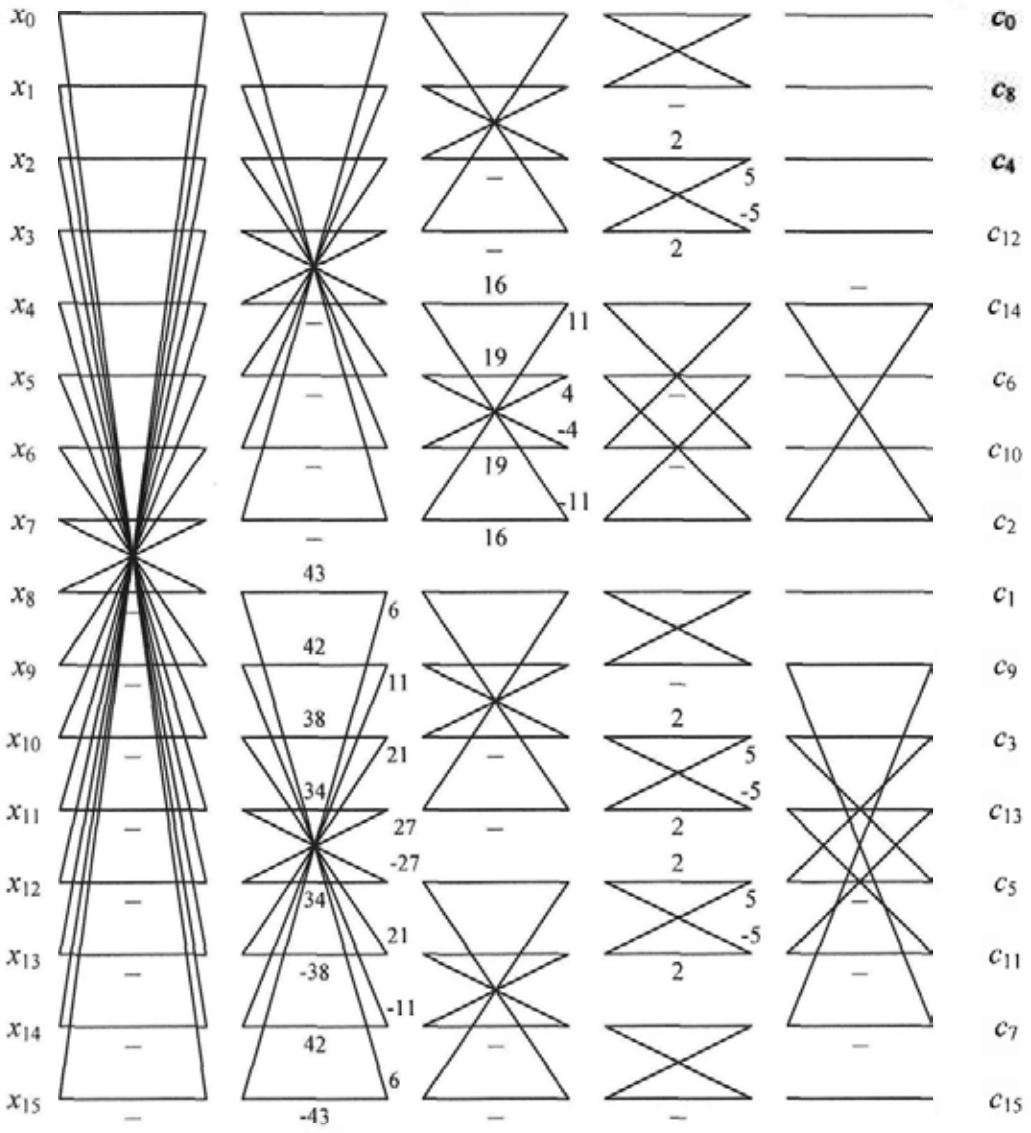


Figure B-3 Data flow of Fast $\mathbf{E}_{Joshi}$.

It is reported that this fast algorithm requires 36 multiplications and 72 additions. In multiplication-free implementation, it requires 64 shifts and 120 additions (total 184 operations).

B.7 Proposed Order-16 Simple Integer Transform

The proposed simple integer transform $\mathbf{T}_{SI}$ is composed of eight order-2 WHTs and two order-8 ICTs. The fast algorithm data flow for the simple integer transform derived from the order-8 transform adopted in H.264/AVC ($\mathbf{T}_{SI\text{-}H264}$) and from the order-8 transform adopted in AVS ($\mathbf{T}_{SI\text{-}AVS}$) are shown in Figure B-4.

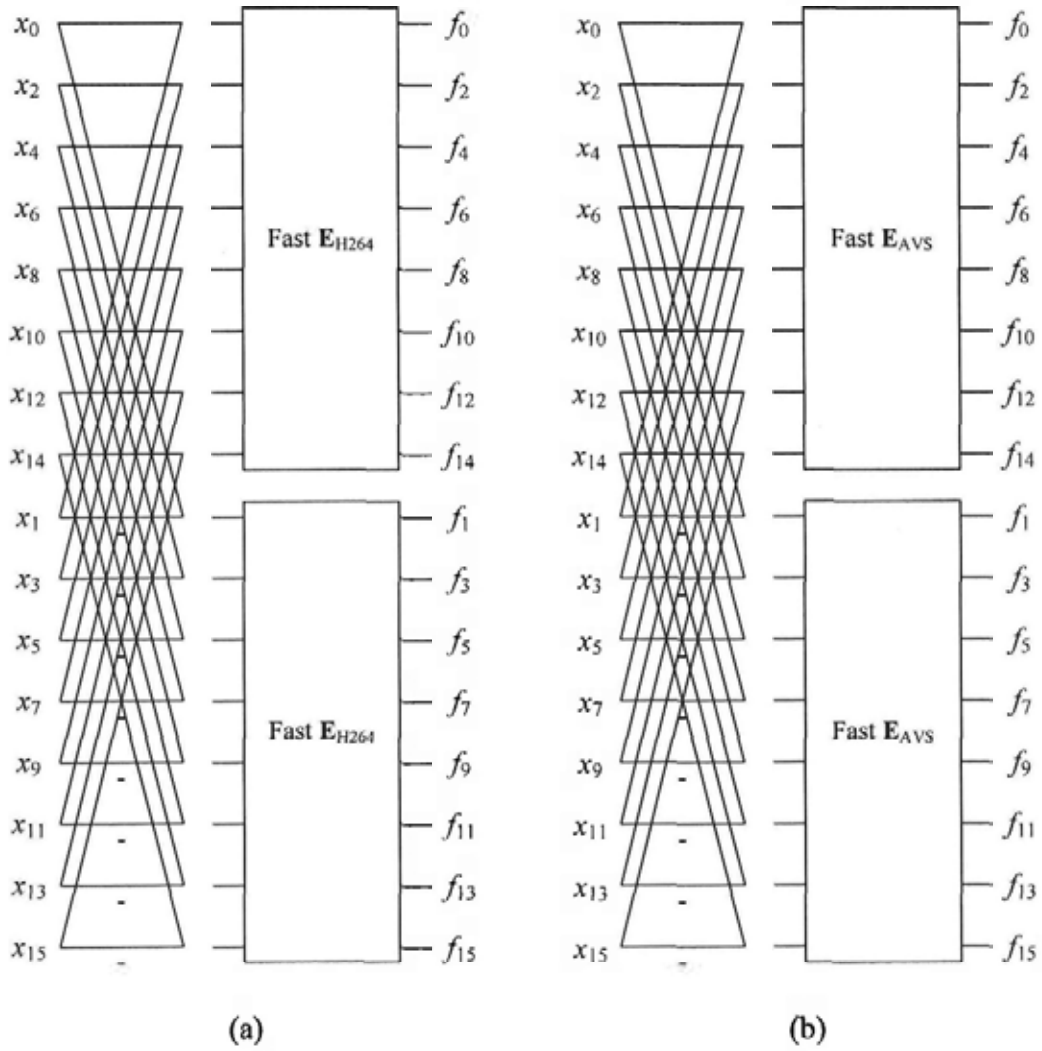


Figure B-4 Data flow of (a) $\mathbf{E}_{SI\text{-}H264}$ and (b) $\mathbf{E}_{SI\text{-}AVS}$.

The order-8 ICT adopted in H.264/AVC requires 14 multiplications and 32 additions while that adopted in AVS requires 14 multiplications and 36 additions. Therefore, the numbers of operations required are:

	Multiplication	Addition
$\mathbf{T}_{SI\text{-}H264}$	28	80
$\mathbf{T}_{SI\text{-}AVS}$	28	88

In multiplication-free implementation, the numbers of operations are:

	Shift	Addition	Total
$\mathbf{T}_{SI\text{-}H264}$	28	80	108
$\mathbf{T}_{SI\text{-}AVS}$	32	92	124

B.8 Proposed Integer Transform from DWWT

Using the method described in Appendix A, the Hybrid Integer Transform $\mathbf{T}_{HI}$ from DWWT can be factorized. Here, $\mathbf{T}_{HI1}$ and $\mathbf{T}_{HI2}$ are taken as examples. Their integer kernels can be factorized as:

$$\mathbf{E}_{HI1} = \begin{bmatrix} \mathbf{E}_{H264} & \mathbf{0}_8 \\ \mathbf{0}_8 & \mathbf{E}_{DWW1} \end{bmatrix} \mathbf{H}_{16} \tag{B-13}$$

$$\mathbf{E}_{HI2} = \begin{bmatrix} \mathbf{E}_{H264} & \mathbf{0}_8 \\ \mathbf{0}_8 & \mathbf{E}_{DWW2} \end{bmatrix} \mathbf{H}_{16} \tag{B-14}$$

Here:

$$\mathbf{E}_{DWW1} = \mathbf{E}_{DWW}(1\,1,1\,1,1\,1,9,8,5,4,1) + \mathbf{E}_{DWW}(0,0,0,0,0,2,0,0)$$

$$= \begin{bmatrix} -1 & & 2 & 1 & & & 2 & -1 \\ -1 & & -2 & & & 2 & 1 & -1 \\ & -1 & 1 & & -1 & 2 & & 2 \\ 2 & 1 & & 2 & 1 & 1 & & \\ & 1 & -1 & & & -1 & 2 & 2 \\ 2 & & & -1 & -2 & & 1 & -1 \\ & 2 & 1 & -2 & 1 & 1 & & \\ -1 & 2 & & 1 & -2 & & -1 & \end{bmatrix} \begin{bmatrix} 1 & & & -1 & 1 & -1 & 1 & \\ & 1 & & 1 & -1 & -1 & 1 & \\ & & 1 & 1 & 1 & 1 & 1 & \\ & & & 1 & 1 & -1 & -1 & -1 \\ 1 & -1 & -1 & 1 & & & & 1 \\ 1 & -1 & 1 & & -1 & & & -1 \\ 1 & 1 & -1 & & & 1 & & -1 \\ 1 & 1 & 1 & & & & -1 & 1 \end{bmatrix}$$

$$\times \begin{bmatrix} 1 & -1 & & & 1 & & 1 & 1 \\ & 1 & -1 & 1 & & & -1 & 1 \\ & 1 & & -1 & -1 & & 1 & 1 \\ & 1 & & 1 & & 1 & & -1 \\ 1 & 1 & 1 & & 1 & -1 & & \\ 1 & & -1 & -1 & 1 & 1 & & \\ 1 & & 1 & & -1 & 1 & -1 & \\ 1 & & -1 & & -1 & -1 & & -1 \end{bmatrix} \begin{bmatrix} 1 & & 1 & & & & & \\ & 1 & & 1 & & & & \\ 1 & & -1 & & & & & \\ & 1 & & -1 & & & & \\ & & & & 1 & & 1 & \\ & & & & & 1 & & 1 \\ & & & & -1 & & 1 & \\ & & & & & -1 & & 1 \end{bmatrix} \tag{B-15}$$

$$+ \begin{bmatrix} & & & & & & & 2 \\ & 2 & & & & & & \\ & & & & & & 2 & \\ & & -2 & & & & & \\ & & & 2 & & & & \\ & & & & -2 & & & \\ & & & & & -2 & & \\ 2 & & & & & & & \end{bmatrix}$$

and

$$\mathbf{E}_{DWW2} = \mathbf{E}_{DWW}(11,11,11,9,8,6,4,1) + \mathbf{E}_{DWW}(0,0,0,0,0,0,1,0)$$

$$= \begin{bmatrix} 1 & 1 & 1 & & 2 & & -3 & -1 \\ & & -1 & 1 & -3 & 1 & -2 & -1 \\ -2 & -1 & & 3 & 1 & -1 & & -1 \\ 1 & -3 & -1 & -1 & & -2 & -1 & \\ & & 2 & 1 & -1 & -1 & -1 & 3 \\ -1 & 2 & & -1 & -1 & -3 & & -1 \\ 3 & 1 & -1 & 2 & & -1 & 1 & \\ -1 & 1 & -3 & & 1 & & -1 & 2 \end{bmatrix} \begin{bmatrix} 2 & & -1 & 1 & & -2 & -1 & \\ & 2 & -1 & 1 & & & 2 & -1 \\ & 2 & 2 & 1 & & & -1 & 1 \\ 2 & 1 & & -2 & -1 & 1 & & \\ 1 & & & & 3 & 1 & & \\ 1 & -1 & & 2 & -1 & 2 & & \\ 1 & -1 & 2 & & & -1 & 2 & \\ & & 1 & & & & -1 & -3 \end{bmatrix} \quad \text{(B-16)}$$

$$\times \begin{bmatrix} 1 & & & & -1 & 1 & & \\ 1 & & & & 1 & & & 1 \\ 1 & & & & & -1 & & -1 \\ & & & & 1 & 1 & & -1 \\ & 1 & 1 & 1 & & & & \\ & 1 & -1 & & & & 1 & \\ & 1 & & -1 & & & -1 & \\ & & 1 & -1 & & & 1 & \end{bmatrix} + \begin{bmatrix} & & & & & & 1 & \\ & & 1 & & & & & \\ & & & & & -1 & & \\ & 1 & & & & & & \\ 1 & & & & & & & \\ & & & & & & & -1 \\ & & & & & 1 & & \\ & & & -1 & & & & \end{bmatrix}$$

The numbers of operations required are:

	Multiplication	Addition
$\mathbf{T}_{HI1}$	**30**	160
$\mathbf{T}_{HI2}$	30	160

In multiplication-free implementation, the numbers of operations are:

	Shift	Addition	Total
$\mathbf{T}_{HI1}$	30	160	190
$\mathbf{T}_{HI2}$	30	158	188

B.9 Proposed LLMICT

Two LLMICTs are taken as the example, A1 and B1.

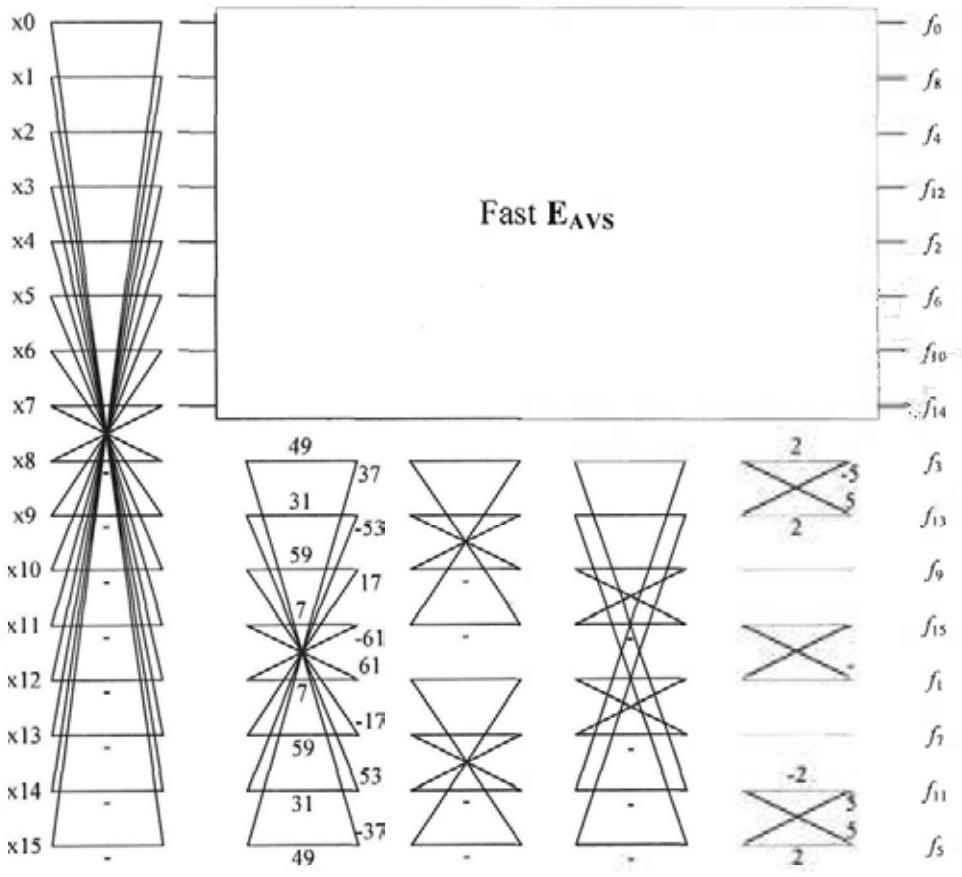


Figure B-5 Data flow of LLMICT A1

This fast algorithm requires 38 multiplications and 78 additions. In multiplication-free implementation, it requires 50 shifts and 110 additions (total 160 operations).

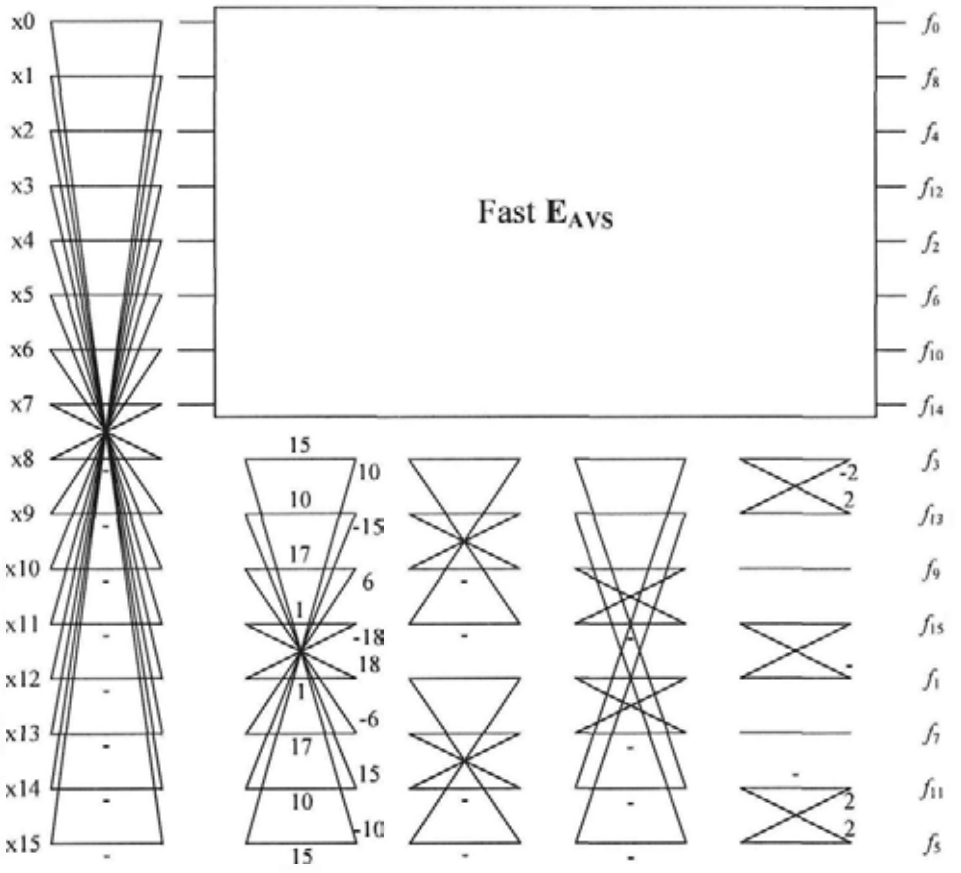


Figure B-6 Data flow of LLMICT B1

This fast algorithm requires 34 multiplications and 78 additions. In multiplication-free implementation, it requires 48 shifts and 92 additions (total 140 operations).

B.10 Reference

[1] Mathias Wien and Shijun Sun, "*ICT Comparison for Adaptive Block Transforms,*" document VCEG-L12, Jan., 2001. [Online] Available: http://wftp3.itu.int/av-arch/video-site/0101_Eib

[2] Bumshik Lee and Munchurl Kim, "*A 16×16 transform kernel with quantization for (ultra) high definition video coding,*" document VCEG-AK13, April 2009. [Online] Available: http://wftp3.itu.int/av-arch/video-site/0904_Yok/

[3] R. Joshi, Y. Reznik, and M. Karczewicz, "*Simplified Transforms for Extended Block Sizes,*" document VCEG-AL30, July 2009. [Online]

Avaliable: http://wftp3.itu.int/av-arch/video-site/0906_LG/

[4] Jie Dong; King Ngi Ngan; Chi-Keung Fong; Wai-Kuen Cham, "*2-D Order-16 Integer Transforms for HD Video Coding,*" IEEE Trans. on CASVT, vol. 19, Issue: 10, pp. 1462 – 1474, Oct. 2009.