IMPLEMENTATION AND EVALUATION OF RESIDUAL COLOR TRANSFORM FOR 4:4:4 LOSSLESS RGB CODING

by

POOJA VASANT AGAWANE

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

August 2008

ACKNOWLEDGEMENTS

I would like to express my heartfelt gratitude to my professor, Dr. K. R. Rao. This work would not have been possible without his erudite vision. He has been a constant source of inspiration all throughout this work. His extremely responsive and attentive approach has helped me improve my work on a regular basis. I thank him for helping me lay my foundation in the world of multimedia processing. It is an honor for me to be a part of his research group.

I would like thank Dr. Davis and Dr. Alavi for taking interest in my work and accepting to be a part of my thesis defense committee. I would like to thank Att, Radhika and Vineeth, my lab mates and the visiting professors in the multimedia processing laboratory for all their help and encouragement. I would like to thank my manager and my team at Intel Corporation, Chandler, Arizona. It is in their company that I gained my first real-world industry exposure.

I dedicate this work to my dad, my mom, my brother and my friend, Jayesh. I thank them for standing by me all throughout my life. This work would not have been possible without them. I take this moment to thank all my friends for being with me through all the times of struggle and celebration.

July 16, 2008

ABSTRACT

IMPLEMENTATION AND EVALUATION OF

RESIDUAL COLOR TRANSFORM FOR

4:4:4 LOSSLESS RGB CODING

Pooja Vasant Agawane, M.S.

The University of Texas at Arlington, 2008

Supervising Professor: Dr. K. R. Rao

The 4:4:4 video sampling format promises an excellent quality video. It is gaining a lot of attention due to its significance in the professional applications of multimedia processing. The use of the RGB color space for video processing is attracting both the industry and the academia. Extensive research is being undertaken to achieve better compression efficiency and high coding gain in the RGB (red, green, blue) color space. In contrast to typical consumer applications, high quality video is required in areas such as professional digital video recording, video post production and digital cinema. These latter applications require all three color components to be represented with identical spatial resolution. Some of these applications require that

each color component of the video signal be captured and displayed with a precision of more than 8 bits per sample.

Most of the time, a video signal is captured and displayed in the RGB color space. During the transition phase between the capture and the display of video, encoding and transmission take place. RGB color space is not an optimum choice for coding and achieving compression. This is because of the significant amount of statistical dependencies between the red, green and blue components of the given video signal. In order to take advantage of these statistical properties, a decorrelating transformation from the RGB color space to some other suitable color space is applied. The various standardization bodies, for example, ITU or SMPTE have defined several color transforms for video coding purposes. One of the color spaces is denoted by YCbCr. This color space includes one luminance component (Y) and two chrominance or color difference components (Cb and Cr). The captured video signal is transformed from the RGB space to YCbCr space. However, this conversion has its limitations. This conversion includes the use of decimal coefficients. Since the samples of a video signal are represented using integers, rounding errors are introduced. Also, in order to achieve high coding efficiency, the complexity of the transform is also increased. To overcome these limitations, the Fidelity Range Extensions (FRExts) amendment of H.264 supports a new color space. This is the YCgCo color space where Y stands for luminance, Cg stands for green chroma and Co stands for orange chroma. The principle of the residual color transform is also introduced in the FRExts. It exploits the redundancy among the residual data of each RGB component after intra/inter prediction.

As a part of this research, the RGB to YCgCo color space transform is applied to the residual data. This thesis aims at the implementation and the evaluation of the residual color transform. This transform is applied to high definition sequences, with the resolution of 1920x1080. The YCgCo color space has improved coding gain relative to both RGB and YCbCr.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

ACRONYMS AND ABBREVIATIONS

AVC: Advanced video coding

AVS: Audio Video Standard

CAVLC: Context adaptive variable length coding

CABAC: Context adaptive binary arithmetic coding

CDF: Cumulative distribution function

DCT: Discrete cosine transform

DVD: Digital video disc or Digital versatile disc

FRExt: Fidelity Range Extensions

HD: High definition

HVS: Human visual system

IEC: International Electrotechnical Commission

ITU-T: International Telecommunication Union – Telecommunication sector

IRCT: Inverse residual color transform

ISO: International Organisation for Standardization

JM: Joint model

JVT: Joint video team

MB: Macroblock

MBAFF: Macroblock adaptive frame/field coding

MC: Motion compensation

ME: Motion estimation

MPEG: Moving picture experts group

MSE: Mean square error

PSNR: Peak signal to noise ratio

RCT: Residual color transform

RGB: Red Green Blue

SEI: Supplemental enhancement information

SMPTE: Society of motion picture and television engineers

TV: Television

VCEG: Video coding experts group

VLC: Variable length coding

WMV: Windows Media Video

YUV: Luminance and chrominance components

CHAPTER 1

INTRODUCTION

1.1 Introduction

Various color spaces are defined to represent a video signal in the digital domain. Some of the most widely used color spaces are YUV and the YCbCr. The YUV color space mainly applies in the analog domain while the YCbCr color space applies to the digital domain. The YCbCr color space consists of one luminance component and two color difference or chrominance components. When all three components are represented at equal resolution, it is termed YUV 4:4:4 sampling [20]. This sampling format is illustrated in Fig. 1.1

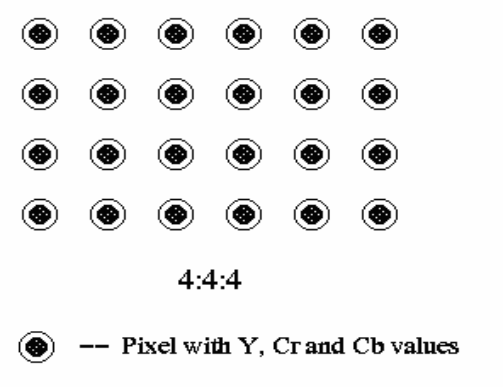

Fig. 1.1 YUV 4:4:4 sampling

The human visual system (HVS) is more sensitive to luminance components than to the chrominance components in a video signal. Hence, the chrominance components can be represented at a lower resolution than the luminance components,

without introducing any significant distortion in the video signal. This is called chroma subsampling [36]. Three types of chroma subsampling are illustrated in the Fig. 1.2.

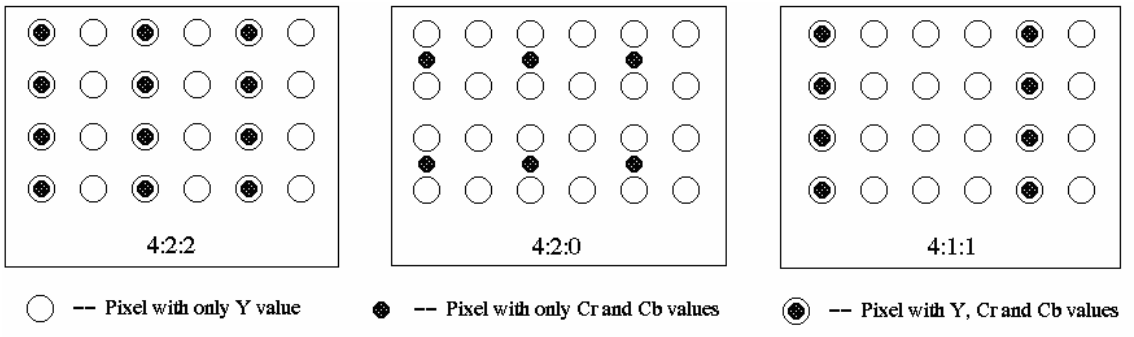

Fig. 1.2 Chroma subsampling types

As shown in Fig. 1.2, for YUV 4:2:2, the chrominance samples are present for every alternate column and for every row, of the luminance sample. Similarly, for YUV 4:2:0, the chrominance samples are present for every alternate row and every alternate column of the luminance samples. For YUV 4:1:1, the chrominance samples are present for every row and every fourth column of the luminance sample. The chroma subsampling helps to achieve a very high compression in the video processing.

Even though chroma subsampling does not introduce any significant perceptual distortion in the output video signal, for high-quality video applications this is not acceptable. These applications demand all three color components to be present with identical spatial resolution. Hence, processing a video signal in the 4:4:4 color space is inevitable in today's world.

Generally, the video is captured and displayed in the RGB domain. However, RGB color space is not an optimum choice for processing the video signal to achieve compression. This is because of the significant amount of statistical dependencies

between the red, green and blue components of the given video signal [8]. In order to take advantage of these statistical properties, the captured video signal is transformed from the RGB space to YCbCr space. However, this conversion includes the use of decimal coefficients. For 8 bits per pixel, the conversion from RGB to YCbCr color space and the reverse transform are achieved using equations 1.1 and 1.2 [24]

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix} + \begin{bmatrix} 65.481 & 128.553 & 24.966 \\ -37.797 & -74.203 & 112.000 \\ 112.000 & -93.786 & -18.214 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{1.1}$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.164 & 1.596 & 0 \\ 1.164 & -0.813 & -0.392 \\ -1.164 & 0 & 2.017 \end{bmatrix} \begin{bmatrix} Y-16 \\ Cr-128 \\ Cb-128 \end{bmatrix} \tag{1.2}$$

Since the samples of a video signal are represented using integers, rounding errors are introduced. Also, in order to achieve high coding efficiency, the complexity of the transform is also increased. To overcome these limitations, the Fidelity Range Extensions (FRExts) [4] amendment of H.264 supports a new color space. This is the YCgCo color space where Y stands for luminance, Cg stands for green chroma and Co stands for orange chroma. The principle of residual color transform [9] is also introduced in the FRExts. It exploits the redundancy among the residual data of each RGB component after intra/inter prediction.

1.2 Thesis outline

The thesis is organized as follows. Chapter 2 describes the overview of the H.264 video coding standard. It explains the various coding tools introduced in H.264. It also explains the encoding and decoding processes in the H.264/AVC standard.

Chapter 3 introduces the fidelity range extensions amendment of the H.264. It describes the motivation for the introduction of the FRExts and explains the new tools of the FRExts. This chapter then explains in detail the YCgCo color space and the principle of residual color transform.

Chapter 4 explains the significance of the lossless coding. It then describes the concept of arithmetic coding. The chapter further explains the lossless coding standards of JPEG-LS and JPEG 2000.

Chapter 5 explains the encoding and the decoding processes in the proposed algorithm. It describes the results obtained using the proposed method. It also gives the results obtained from JM software, independent JPEG, JPEG-2000 and JPEG-LS simulations.

Chapter 6 outlines the conclusions of this research. Future work for this research is suggested.

CHAPTER 2

OVERVIEW OF H.264

2.1 Introduction

H.264/MPEG4-Part 10 advanced video coding (AVC) is one of the latest video coding standards introduced in the world of video compression [1]. The H.264 standard was developed by the Joint Video Team (JVT), consisting of VCEG (Video Coding Experts Group) of ITU-T (International Telecommunication Union – Telecommunication standardization sector), and MPEG (Moving Picture Experts Group) of ISO/IEC [1].

This new coding standard is noted for enhanced compression efficiency. It can support the various interactive (video telephony) and non-interactive applications (broadcast, streaming, storage, video on demand) since it facilitates a network friendly video representation [2]. The previous coding standards, MPEG-1, MPEG-2, MPEG-4 part 2, H.261, H.262 and H.263 [1] [37] are the basis on which the H.264 is developed. It uses the following basic principles of video compression:

- Transform for reduction of spatial correlation
- Quantization for control of bitrate
- Motion compensated prediction for reduction of temporal correlation
- Entropy coding for reduction in statistical correlation.

H.264 has improved coding efficiency as the functional elements of encoder and decoder are modified to include additional coding tools. These new features added to the H.264 standard are listed as follows:

- Adaptive intra-picture prediction
- Small block size transform with integer precision
- Multiple reference pictures and generalized B-frames
- Variable block sizes
- Quarter pel precision for motion compensation
- Content adaptive in-loop deblocking filter and
- Improved entropy coding by introduction of CABAC (context adaptive binary arithmetic coding) and CAVLC (context adaptive variable length coding)

The increase in the coding efficiency and increase in the compression ratio results to a greater complexity of the encoder and the decoder algorithms of H.264, as compared to previous coding standards. In order to develop error resilience for transmission of information over the network, H.264 supports the following techniques:

- Flexible macroblock ordering
- Switched slice
- Arbitrary slice order
- Redundant slice
- Data partitioning
- Parameter setting

2.2 Profiles and levels of H.264

The H.264/AVC standard is composed of a wide range of coding tools. Also, the standard addresses a large range of bit rates, resolutions, qualities, applications and services. Not all the tools and all the bitrates are required for any given application at a given point of time. All the various tools of H.264 are grouped in profiles. Profiles are defined as a subset of coding tools. They help to maximize the interoperability while limiting the complexity [5]. Also, the various levels define the various parameters like size of decoded pictures, bit rate, etc.

The profiles defined for H.264 can be listed as follows:

1. Baseline profile

2. Extended profile

3. Main profile

Fig. 2.1 illustrates the coding tools for the various profiles of H.264.

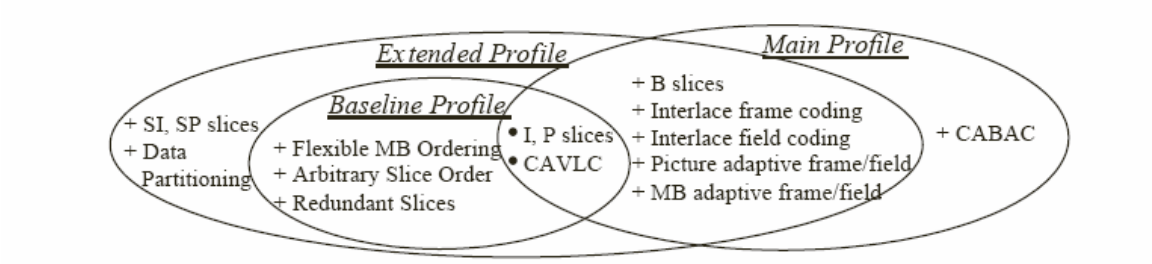

Fig. 2.1: Profile structure of H.264 [5]

Baseline profile: The list of tools included in the baseline profile are I (intra coded) and P (predictive coded) slice coding, enhanced error resilience tools of flexible macroblock ordering, arbitary slices and redundant slices. It also supports CAVLC (context-based adaptive variable length coding). The baseline profile is intended to be

used in low delay applications, applications demanding low processing power, and in high packet loss environments. This profile has the least coding efficiency among all the three profiles.

Main profile: The coding tools included in the main profile are I, P, and B (bidirectionally prediction coded) slices, interlace coding, CAVLC and CABAC (context-based adaptive binary arithmetic coding). The tools not supported by main profile are error resilience tools, data partitioning and SI (switched intra coded) and SP (switched predictive coded) slices. This profile is aimed to achieve highest possible coding efficiency.

Extended profile: This profile has all the tools included in the baseline profile. As illustrated in the Fig. 2.1, this profile also includes B, SP and SI slices, data partitioning, interlace frame and field coding, picture adaptive frame/field coding and MB adaptive frame/field coding. This profile provides better coding efficiency than baseline profile. The additional tools result in increased complexity.

In H.264 /AVC, 16 levels are specified. Each level defines upper bounds for the bit stream or lower bounds for the decoder capabilities. A profile and level can be combined to define the conformance points. These points signify the point of inter-operability for applications with similar functional requirements [6]. The levels defined in H.264 are listed in Table 2.1. The level ‘1b’ was added in the FRExts amendment.

Table 2.1 Levels defined in H.264 [7]

Level Number	Typical Picture Size	Typical frame rate	Maximum compressed bit rate (for VCL) in Non-FRExt profiles	Maximum number of reference frames for typical picture size
1	QCIF	15	64 kbps	4
1b	QCIF	15	128 kbps	4
1.1	CIF or QCIF	7.5 (CIF) / 30 (QCIF)	192 kbps	2 (CIF) / 9 (QCIF)
1.2	CIF	15	384 kbps	6
1.3	CIF	30	768 kbps	6
2	CIF	30	2 Mbps	6
2.1	HHR (480i or 576i)	30 / 25	4 Mbps	6
2.2	SD	15	4 Mbps	5
3	SD	30 / 25	10 Mbps	5
3.1	1280x720p	30	14 Mbps	5
3.2	1280x720p	60	20 Mbps	4
4	HD Formats (720p or 1080i)	60p / 30i	20 Mbps	4
4.1	HD Formats (720p or 1080i)	60p / 30i	50 Mbps	4
4.2	1920x1080p	60p	50 Mbps	4
5	2kx1k	72	135 Mbps	5
5.1	2kx1k or 4kx2k	120 / 30	240 Mbps	5

2.3 H.264 encoder

Fig. 2.2 illustrates the schematic of the H.264 encoder. The operation of the H.264 encoder is briefly explained as follows. A given image is split into blocks which can be varied. The encoder may perform intra-coding or inter-coding for the macroblocks of a given picture. Intra coded frames are encoded and decoded independently. They do not need any reference frames. Hence they provide access points to the coded sequence where decoding can start. It employs nine spatial prediction modes which reduce spatial redundancy in the source signal of the picture. These prediction modes are explained in section *2.3.1*. Inter-coding uses inter-prediction of a given block from some previously decoded pictures. It uses motion vectors for block-based inter-prediction and results in reduction of temporal redundancy

among different pictures. The prediction residual signals then undergo transformation to remove spatial correlation in the block. The transformed coefficients, thus obtained, undergo quantization.

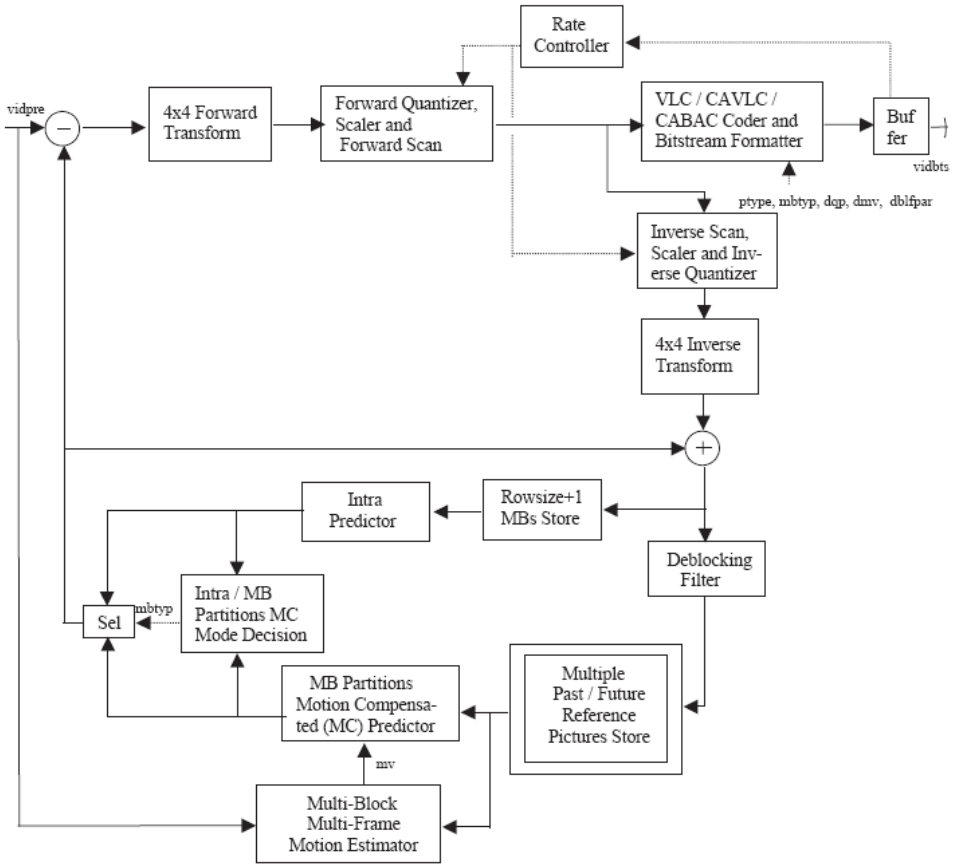

Fig. 2.2: Block diagram of H.264 encoder [1]

The motion vectors, obtained from inter-prediction or intra-prediction modes are combined with the quantized transform coefficient information. They are then encoded using entropy code such as context-based adaptive variable length coding (CAVLC) or context-based adaptive binary arithmetic coding (CABAC) [1]. There is a local decoder within the H.264 encoder. This local decoder performs the operations of inverse quantization and inverse transform to obtain the residual signal in the spatial domain.

The prediction signal is added to the residual signal to reconstruct the input frame. This input frame is fed in the deblocking filter to remove blocking artifacts at the block boundaries. The output of the deblocking filter is then fed to inter/intra prediction blocks to generate prediction signals.

The various coding tools used in the H.264 encoder are explained below.

2.3.1. Intra prediction

In intra prediction, the samples of the macroblock are predicted using the macroblock of the same image. For the luminance component, there are two types of prediction schemes implemented. These two schemes can be referred as INTRA_4x4 and INTRA_16x16 [6]. In INTRA_4x4, a macroblock of size 16x16 samples is divided into 16 4x4 subblocks. Intra prediction scheme is applied individually to these 4x4 subblocks. There are nine different prediction modes supported (Fig. 2.3).

In mode 0, the samples of the macroblock are predicted from the neighboring samples on the top. In mode 1, the samples of the macroblock are predicted from the neighboring samples from the left. In mode 2, the mean of all the neighboring samples is used for prediction. Mode 3 is in diagonally down-left direction. Mode 4 is in diagonal down-right direction. Mode 5 is in vertical-right direction. Mode 6 is in horizontal-down direction. Mode 7 is in vertical-left direction. Mode 8 is in horizontal-up direction. The predicted samples are calculated from a weighted average of the prediction samples A to M.

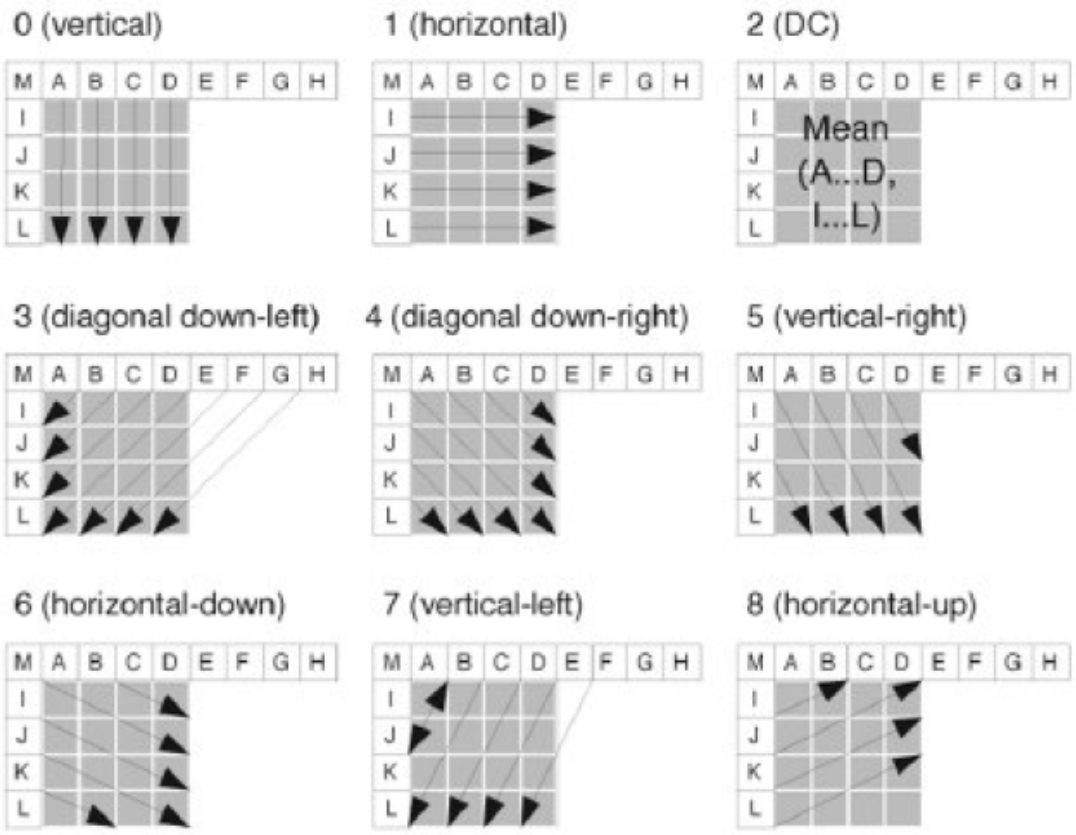

Fig. 2.3 Nine prediction modes for intra-prediction [32]

For prediction of 16x16 intra prediction of luminance components, four modes are used. The three modes of mode 0 (vertical), mode 1 (horizontal) and mode 2 (DC) are similar to the prediction modes for 4x4 block. In the fourth mode, the linear plane function is fitted in the neighboring samples. The chroma macroblock is predicted from neighboring chroma samples. The four prediction modes used for the chroma blocks are similar to 16x16 luma prediction modes. The number in which the prediction modes are ordered is different for chroma macroblock: mode 0 is DC, mode 1 is horizontal, mode 2 is vertical and mode 3 is plane. The block sizes for the chroma prediction depend on

the sampling format. For 4:2:0 format, 8x8 size of chroma block is selected. For 4:2:2 format, 8x16 size of chroma block is selected. For 4:4:4 format, 16x16 size of chroma block is selected [1].

2.3.2 Inter prediction

There exists temporal correlation among the images in a video sequence. This correlation is reduced by inter prediction through the use of motion estimation and compensation algorithms [1]. The current image is partitioned into macroblocks or smaller blocks. A 16x16 macroblock can be partitioned in 16x16, 16x8, 8x16, 8x8 sized blocks. A 8x8 sub-macroblock can be further partitioned in 8x4, 4x8, 4x4 sized blocks. Fig. 2.4 illustrates the partitioning of a macroblock and a sub-macroblock [6]. The input video characteristics govern the block size. The smaller the size of the block, the larger the number of bits required to encode the motion vectors. The extra data needs to be sent for the type of partition. However, the motion compensation residual data is reduced.

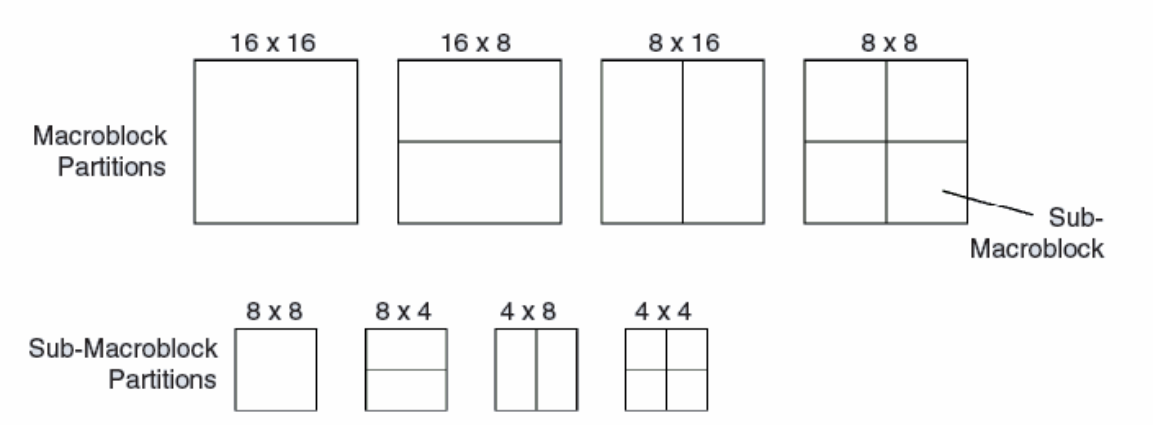

Fig. 2.4 Partitioning of macroblock and sub-macroblock for inter prediction [6]

The reference pictures used for inter prediction are previously decoded frames and are stored in the picture buffer. H.264 supports the use of multiple frames as reference frames. This is implemented by the use of an additional picture reference parameter which is transmitted along with the motion vector. The parameters d_t in Fig. 2.5 are the image reference parameters.

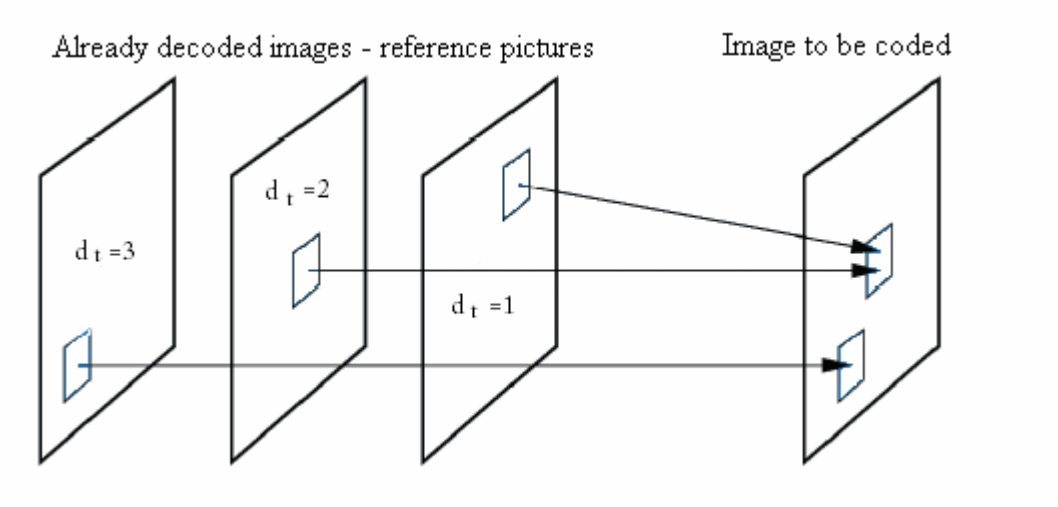

Fig. 2.5 Motion compensated prediction with multiple reference images [6]

2.3.3 Transform coding

There is high spatial redundancy among the prediction error signals. H.264 implements a block-based transform to reduce this spatial redundancy [1]. The former standards of MPEG-1 and MPEG-2 employed a two dimensional discrete cosine transform (DCT) [23] for the purpose of transform coding of the size 8x8 [6]. H.264 uses integer transforms instead of the DCT. The size of these transforms is 4x4 [6].

The advantages of using a smaller block size in H.264 are stated as follows:

- The reduction in the transform size enables the encoder to better adapt the prediction error coding to the boundaries of the moving objects and to match the transform block size with the smallest block size of motion compensation.
- The smaller block size of the transform leads to a significant reduction in the ringing artifacts.
- The 4x4 transform has benefit for removing the need for multiplications.

H.264 employs a hierarchical transform structure, in which the DC coefficients of neighboring 4x4 transforms for luma signals are grouped into 4x4 blocks and transformed again by the Hadamard transform. As shown in Fig. 2.6, the first transform (matrix H_1) is applied to all samples of all prediction error blocks of the luminance component (Y) and for all blocks of chrominance components (Cb and Cr). For blocks with mostly flat pixel values, there is significant correlation among transform DC coefficients of neighboring blocks. Hence, the standard specifies the 4x4 Hadamard transform (matrix H_2 in Fig. 2.6) for luma DC coefficients for 16x16 intra-mode only, and 2x2 Hadamard transform (matrix H_3 in Fig. 2.6) for chroma DC coefficients.

$$H_1 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \qquad H_2 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \qquad H_3 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Fig. 2.6 Matrices H_1, H_2 and H_3 of the three transforms applied in H.264 [6]

2.3.4 Deblocking filter

H.264 employs a block-based transform in intra-prediction and inter-prediction coding. The transform coefficients then undergo quantization. These two steps result in blocking artifacts. These artifacts are reduced by the in-loop deblocking filter of H.264. It reduces the artifacts at the block boundaries and prevents the propagation of the accumulated noise. However, the implementation of this filter results to an increase in the implementation complexity. Fig. 2.7 illustrates a macroblock with sixteen 4x4 sub-blocks along with their boundaries.

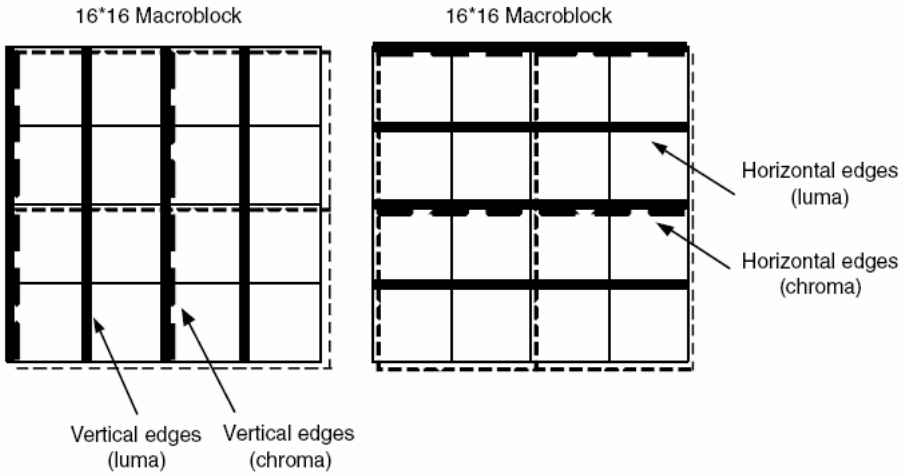

Fig. 2.7: Boundaries in a macroblock to be filtered (luma boundaries shown with solid lines and chroma boundaries shown with dotted lines) [1]

As shown in the Fig. 2.7 the luma deblocking filter process is performed on the 16 sample edges – shown by solid lines. The chroma deblocking filter process is performed on 8 sample edges – shown in dotted lines.

H.264 employs deblocking process adaptively at the following three levels:

- At slice level – global filtering strength is adjusted to the individual characteristics of the video sequence

- At block-edge level – deblocking filter decision is based on inter or intra prediction of the block, motion differences and presence of coded residuals in the two participating blocks.
- At sample level – it is important to distinguish between the blocking artifact and the true edges of the image. True edges should not be deblocked. Hence decision for deblocking at an sample level becomes important.

2.3.5 Entropy coding

H.264 uses variable length coding to match a symbol to a code based on the context characteristics. All the syntax elements except for the residual data are encoded by the Exp-Golomb codes [1]. The residual data is encoded using CAVLC. The main and the high profiles of H.264 use CABAC.

- Context-based adaptive variable length coding (CAVLC):

After undergoing transform and quantization the probability that the level of coefficients is zero or +1 is very high [1]. CAVLC handles these values differently. It codes the number of zeroes and +1. For other values, their values are coded.

- Context-based adaptive binary arithmetic coding (CABAC):

This technique utilizes the arithmetic encoding [32] to achieve good compression. The schematic for CABAC is shown in Fig.. 2.8.

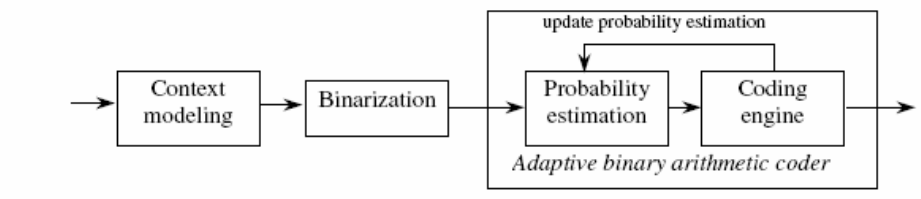

Fig. 2.8 Schematic block diagram of CABAC [1]

CABAC consists of three steps:

Step 1: Binarization: A non-binary value is uniquely mapped to a binary sequence

Step 2: Context modeling: A context model is a probability model for one or more elements of binarized symbol. The probability model is selected such that corresponding choice may depend on previously encoded syntax elements.

Step 3: Binary arithmetic coding: An arithmetic encoder encodes each element according to the selected probability model.

2.3.6 B-slices and adaptive weighted prediction

Temporal correlation can be efficiently reduced by bidirectional prediction. It uses multiple reference pictures. The standards, before H.264, with B pictures use the bidirectional mode, with limitation that it allows the combination of a previous and subsequent prediction signals. In the previous standards, one prediction signal is derived from subsequent inter-picture, another from a previous picture, the other from a linear averaged signal of two motion compensated prediction signals. H.264 supports forward/backward prediction pair and also supports forward/forward and backward/backward prediction pair [1]. Considering two forward references for prediction is beneficial for motion compensated prediction of a region just before scene change. Considering two backward reference frames is beneficial for frames just after scene change. H.264 also allows bi-directionally predictive-coded slice may also be used as references for inter-coding of other pictures. Except H.264, all the existing standards consider equal weights for reference pictures. Equal weights of reference signals are averaged and the prediction signal is obtained. H.264 uses weighted

prediction [1]. It can be used for a macroblock of P slice or B slice. Different weights are assigned can be assigned to two different reference signals and the prediction signal is calculated as follows:

$$p = w1 * r1 + w2 * r2 \qquad (2.1)$$

where p is the prediction signal, r1 and r2 are the reference signals and w1 and w2 are the prediction weights.

2.4 H.264 decoder

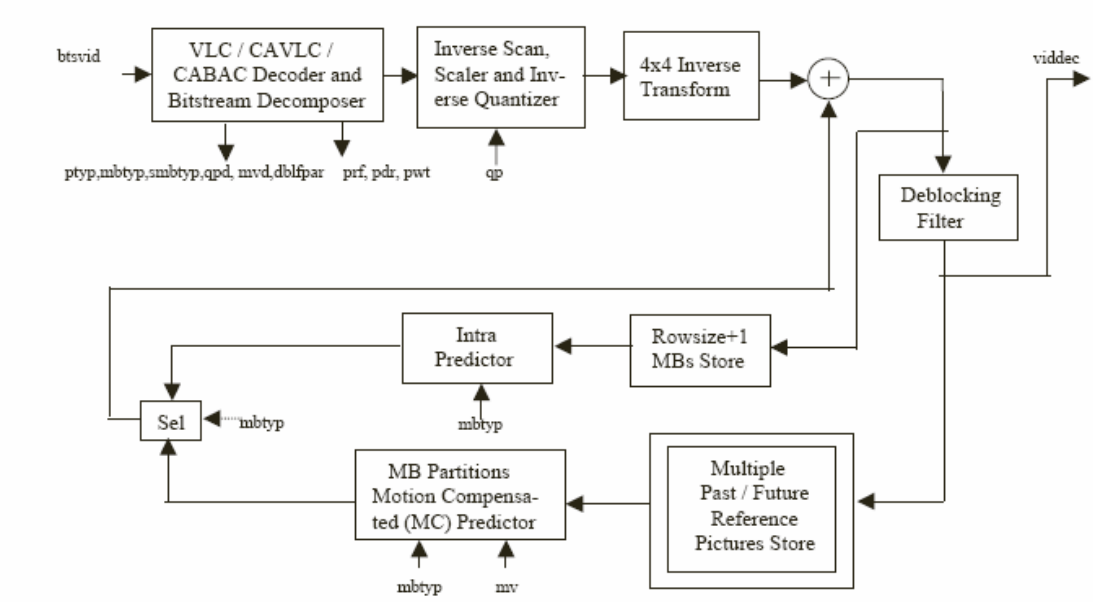

Fig. 2.9: H.264 decoder block diagram [5]

The H.264 decoder works similar in operation to the local decoder of H.264 encoder. An encoded bit stream is the input to the decoder. Entropy decoding (CABAC or CAVLC) takes place on the bit stream to obtain the transform coefficients. These coefficients are then inverse scanned and inverse quantized. This gives residual block data in the transform domain. Inverse transform is performed to obtain the data in the

pixel domain. The resulting output is 4x4 blocks of residual signal. Depending on inter-predicted or intra-predicted, an appropriate prediction signal is added to the residual signal. For an inter-coded block, a prediction block is constructed depending on the motion vectors, reference frames and previously decoded pictures. This prediction block is added to the residual block to reconstruct the video frames. These reconstructed frames then undergo deblocking before they are stored for future use for prediction or being displayed.

2.5 Comparison of H.264 with WMV9 and AVS China [1]

Table 2.2 Comparison of H.264/MPEG4 part 10 with WMV-9 and AVS [1]

Feature/standard	MPEG-4 part 10/H.264	WMV-9	AVS
Prediction block size	16×16, 8×16, 16×8, 8×8, 4×8, 8×4, 4×4	16×16, 8×8	16×16, 8×8
Intra-prediction	4×4, 8×8: 9 modes 16×16: 4 modes	No	8×8: 5 modes
Transform	8×8, 4×4 integer DCT 4×4, 2×2 Hadamard	8×8, 8×4, 4×8, 4×4 integer DCT	Asymmetric 8×8 integer DCT
Quantization	Scalar quantization	Dead zone, uniform scalar quantization	Scalar quantization
Entropy coding	VLC, CAVLC, CABAC	Multiple VLC tables	VLC
Sub-pel filter	1/2-pel: 6-tap, 1/4-pel: 2-tap	1/2-pel: 4-tap, 1/4-pel: 4-tap	1/2-pel: 4-tap, 1/4-pel: 4-tap
Reference picture	Multiple pictures	One picture	Two pictures
Bidirectional prediction mode	Forward/backward, forward/forward, backward/backward, 2 motion vectors	Forward/backward, 2 motion vectors	Forward/backward, symmetric 1 motion vector
Weighted prediction	Yes	Yes	Yes
Deblocking filter	Yes	Yes	Yes

The standards of WMV-9 [25] and AVS China [26] have developed encoder and decoder algorithms similar to H.264. WMV-9 is adopted by SMPTE as VC1 [32]. WMV-9 implements adaptive block size transform. It allows 8x8 blocks to be encoded using either one 8x8 transform, two horizontally stacked 8x4 transforms, two vertically

stacked 4x8 transforms or four 4x4 transforms. It also uses integer transforms, but the transform matrices vary from the ones used in H.264. The 8x8 matrix for inverse transform is shown as follows [1]:

$$\begin{bmatrix} 12 & 12 & 12 & 12 & 12 & 12 & 12 & 12 \\ 16 & 15 & 9 & 4 & -4 & -9 & -15 & -16 \\ 16 & 6 & -6 & -16 & -16 & -6 & 6 & 16 \\ 15 & -4 & -16 & -9 & 9 & 16 & 4 & -15 \\ 12 & -12 & -12 & 12 & 12 & -12 & -12 & 12 \\ 9 & -16 & 4 & 15 & -15 & -4 & 16 & 9 \\ 6 & -16 & 16 & -6 & -6 & 16 & -16 & 6 \\ 4 & -9 & 15 & -16 & 16 & -15 & 9 & -4 \end{bmatrix}$$

Also, a 4x4 inverse transform is shown as follows [1]:

$$\begin{bmatrix} 17 & 17 & 17 & 17 \\ 22 & 10 & -10 & -22 \\ 17 & -17 & -17 & 17 \\ 10 & -22 & 22 & -10 \end{bmatrix}$$

It uses simple variable length codes for entropy coding, but allows the use of multiple code tables.

The main application of AVS is for broadcast TV, HD-DVD and broadband video networking and mobile networks. The transform size used in AVS is only 8x8. it uses one of the 5 modes for intra prediction. The five prediction modes supported by AVS are mode 0 – vertical, mode 1 – horizontal, mode 2 – DC, mode 3 – diagonal down left and mode 4 – diagonal down right. The DC is obtained after lowpass filtering. AVS implements separable, inter-precise, 8x8 DCT and asymmetric transform in which both post-scaling and pre-scaling are involved in the encoder side. A linear scalar

quantizer is used for quantization. All the syntax elements, including the transform coefficients are encoded using Exp-Golomb codes [21]. The matrix for 8x8 inverse transform is shown as follows [1]:

$$\begin{bmatrix} 8 & 10 & 10 & 9 & 8 & 6 & 4 & 2 \\ 8 & 9 & 4 & -2 & -8 & -10 & -10 & -6 \\ 8 & 6 & -4 & -10 & -8 & 2 & 10 & 9 \\ 8 & 2 & -10 & -6 & 8 & 9 & -4 & -10 \\ 8 & -2 & -10 & 6 & 8 & -9 & -4 & 10 \\ 8 & -6 & -4 & 10 & -8 & -2 & 10 & -9 \\ 8 & -9 & 4 & 2 & -8 & 10 & -10 & 6 \\ 8 & -10 & 10 & -9 & 8 & -6 & 4 & -2 \end{bmatrix}$$

The three coding standards of H.264, WMV-9 and AVS have similar functionalities with some changes in the adaptive/non-adaptive transform sizes, entropy coding, directional prediction, deblocking filter, etc.

2.6 Summary

This chapter gives an overview of the H.264/MPEG4-Part 10 AVC standard. This is the latest coding standard in the field of multimedia processing and has a wide range of applications, in various areas. The basic algorithm for encoding and decoding video signals in the H.264 format is based on the previous standards such as MPEG-1, MPEG-2, H.261, etc. H.264 standard brings with it numerous new and advanced coding tools like in-loop deblocking filter, integer transform, multiple reference frames, etc. This chapter explains the various coding tools, which are a part of H.264 standard. The chapter also briefly compares H.264 with other coding standards of WMV-9 and AVS.

The next chapter introduces the fidelity range extensions amendment to H.264. This amendment was aimed at gaining better coding efficiency for high quality video applications. Various new coding tools were introduced as a part of FRExts. The chapter explains the YCgCo color space in detail and the concept of residual color transform.

CHAPTER 3

FIDELITY RANGE EXTENSIONS AND RESIDUAL COLOR TRANSFORM

3.1 Introduction

The first amendment of H.264/MPEG-4 AVC video coding standard was completed in July 2004 and the corresponding final draft amendment text was released in September 2004 [4]. A new set of coding tools were introduced as a part of this amendment. These are termed as "Fidelity Range Extensions" (FRExts). They are aimed at achieving significant improvements in the coding efficiency for higher fidelity video material. The application areas for the FRExt tools are professional film production, video post production and high-definition TV/DVD.

3.2 Coding tools

Various tools are introduced in the so called Fidelity Range Extensions but the main difference between FRExt and non-FRExt H.264 codec is the use of 8x8 transform in addition to 4x4 transforms. The various tools are explained below:

3.2.1 8x8 Intra spatial prediction

H.264 implements block based intra prediction where spatially neighboring samples of a block are used for the prediction of the current block. The two types of prediction introduced in H.264 for luma samples are INTRA_16x16 – prediction for the entire macroblock and INTRA_4x4 – prediction for sixteen 4x4 blocks within a macroblock. Based on 4x4 prediction, an intermediate prediction block size of 8x8 was

introduced for spatial luma prediction in the FRExts. Fig. 3.1 (a to i) illustrate the 8x8 block of the luma samples and its neighboring samples, along with the 9 modes of prediction.

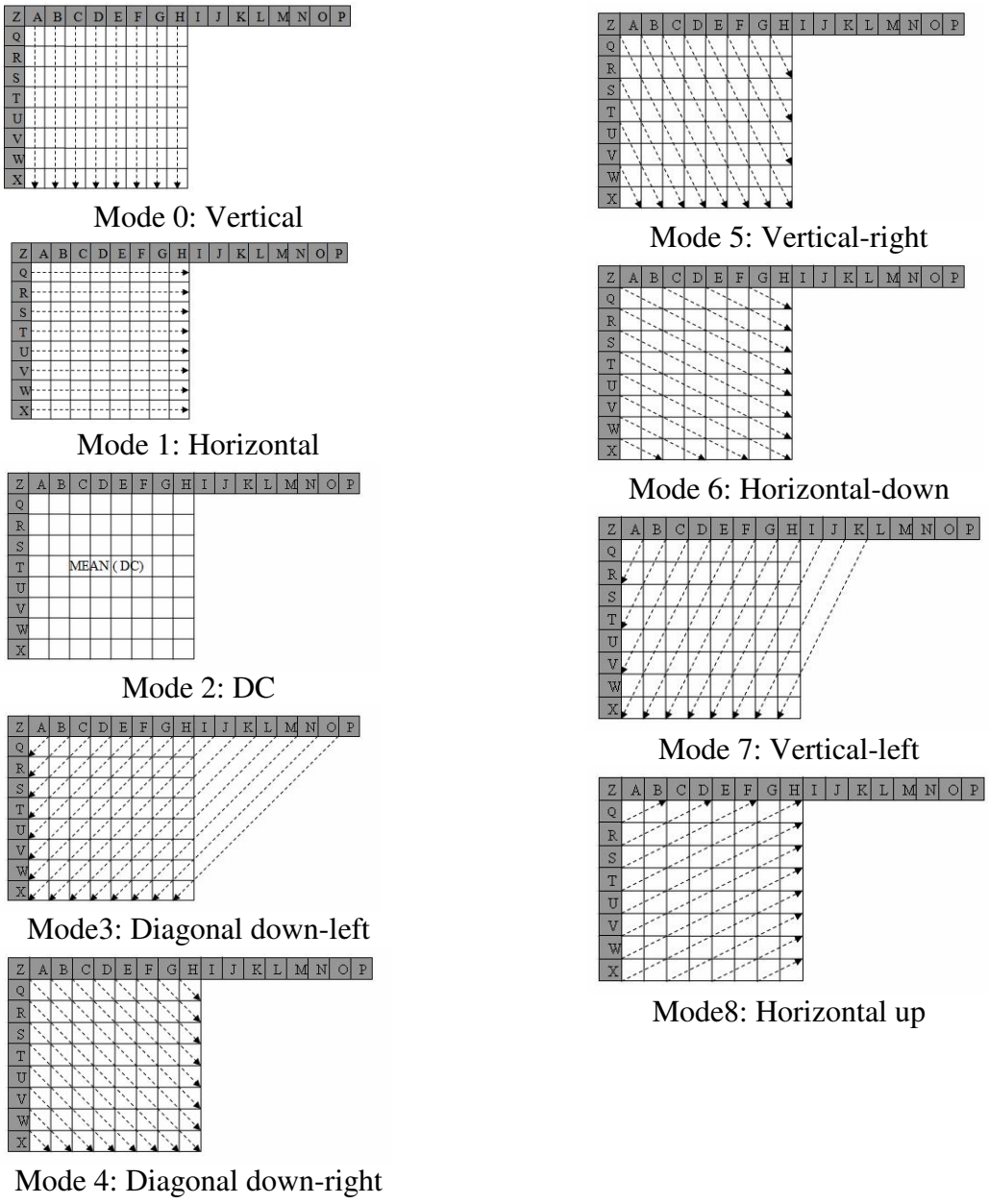

Fig. 3.1 Nine prediction modes used for 8x8 spatial luma prediction

The encoder selects the size of the prediction block – 4x4 or 8x8 or 16x16, depending on the application.

3.2.2 8x8 Transform

The first version of H.264/MPEG4 AVC uses two different 4x4 transforms for the coding of the luma prediction error signal [4]. The use of small block-size transforms in H.264 significantly reduces ringing artifacts, in addition to reducing the computational complexity. High-fidelity video demands preservation of fine details and textures. This requires large basis functions. As a tradeoff between the reduction of ringing artifacts and preservation of fine details, an intermediate transform of 8x8 size is introduced in FRExts. FRExts allow the encoder to choose adaptively between the 4x4 and 8x8 transform for luma samples on a macroblock level. The two-dimensional 8x8 transform in the FRExts is specified in a separable way as a one-dimensional horizontal transform followed by a one-dimensional vertical transform. The corresponding one-dimensional transform is given by the (non-normalized) transformation matrix 8x8, stated as follows [4]:

$$T_{8x8} = \begin{bmatrix} 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 \\ 12 & 10 & 6 & 3 & -3 & -6 & -10 & -12 \\ 8 & 4 & -4 & -8 & -8 & -4 & 4 & 8 \\ 10 & -3 & -12 & -6 & 6 & 12 & 3 & -10 \\ 8 & -8 & -8 & 8 & 8 & -8 & -8 & 8 \\ 6 & -12 & 3 & 10 & -10 & -3 & 12 & -6 \\ 4 & -8 & 8 & -4 & -4 & 8 & -8 & 4 \\ 3 & -6 & 10 & -12 & 12 & -10 & 6 & -3 \end{bmatrix}$$

As mentioned, the encoder adaptively chooses between 4x4 and 8x8 transforms. The transform size selection process is limited by the following condition:

- If an inter-coded macroblock has a sub-partition smaller than 8x8 (i.e. 4x8, 8x4 or 4x4), then 4x4 transform has to be used.
- If an intra-coded macroblock is predicted using 8x8 luma spatial prediction, only then 8x8 transform is used.

FRExts suggest default perceptual weighting matrices for 4x4 and 8x8 integer DCT coefficients. Scaling matrix reflecting visual perception is simply a multiplier applied during the inverse quantization. Perceptual scaling matrices can be designed and customized at the encoder. Hence, as they are not default HVS matrices, these matrices need to be transmitted to the decoder at the sequence or picture level. The default scaling matrix for 8x8 integer DCT is shown below [1].

$$\begin{bmatrix} 6 & 10 & 13 & 16 & 18 & 23 & 25 & 27 \\ 10 & 11 & 16 & 18 & 23 & 25 & 27 & 29 \\ 13 & 16 & 18 & 23 & 25 & 27 & 29 & 31 \\ 16 & 18 & 23 & 25 & 27 & 29 & 31 & 33 \\ 18 & 23 & 25 & 27 & 29 & 31 & 33 & 36 \\ 23 & 25 & 27 & 29 & 31 & 33 & 36 & 38 \\ 25 & 27 & 29 & 31 & 33 & 36 & 38 & 40 \\ 27 & 29 & 31 & 33 & 36 & 38 & 40 & 42 \end{bmatrix}$$

In FRExts two scans similar to 4x4 transform switched for frame/field coding are shown. Coefficient scanning is based on the decreasing variances and to maximize number of zero-valued coefficients along the scan.

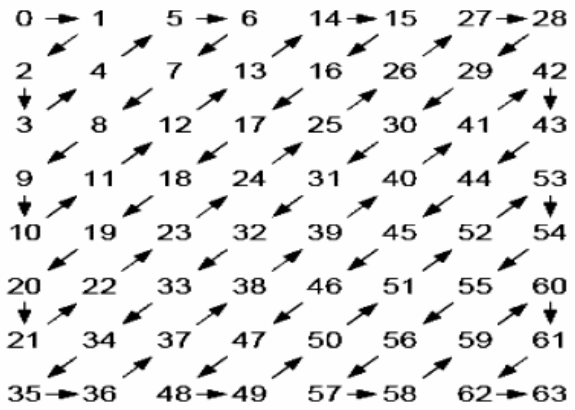

Fig. 3.2 Zig-zag frame scan for 8x8 block [1]

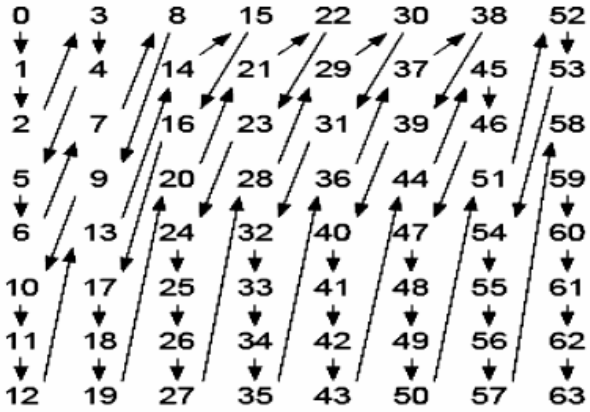

Fig. 3.3 Field scan for 8x8 block [1]

Scanning order of quantized 4x4 integer DCT coefficients is shown in Fig. 3.4 and Fig. 3.5.

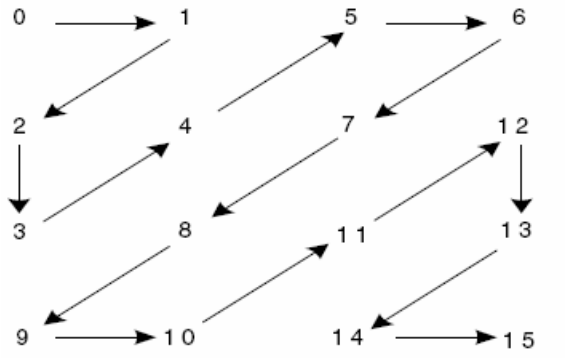

Fig. 3.4 Zig-zag scan for 4x4 block [1]

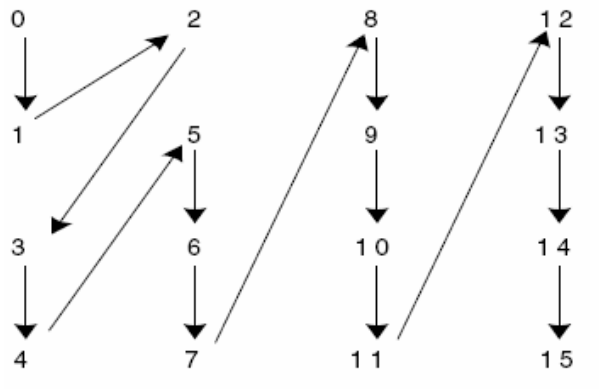

Fig. 3.5 Alternate scan for 4x4 block [1]

3.2.3 More coding tools

The FRExt amendment supports extended sample bit depth as well as 4:2:2 and 4:4:4 chroma formats (Fig. 1.1 and Fig. 1.2). For this, the following coding tools are introduced:

- Encoder-specified perceptual-based quantization scaling matrices.

  The encoder can specify a matrix for scaling factor according to the specific frequency associated with the transform coefficient for use in inverse quantization scaling by the decoder. This allows optimization of the subjective quality according to the sensitivity of the human visual system, less sensitive to the coded error in high frequency transform coefficients [1].

- A residual color transform consisting of a reversible integer-based color conversion from (4:4:4) RGB to YCgCo color space applied to residual data only.

- An efficient lossless representation of the video with a simple bypass of transform and quantization.

- New supplemental enhancement information (SEI) messages for enabling enhancements of decoded video

3.3 High profiles

The FREext amendment has introduced four new profiles to H.264 suite of profiles. These are called as High profiles. These profiles are illustrated in the Fig. 3.6

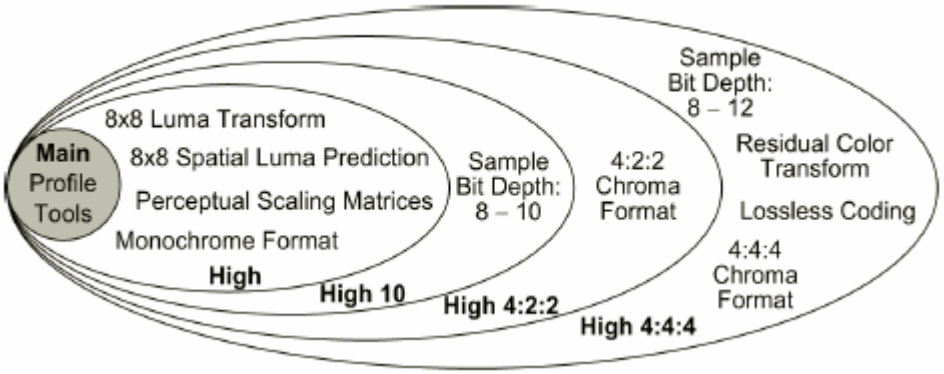

Fig. 3.6: High profiles introduced in FREext amendment [4]

These four high profiles are briefly explained below:

- High profile – supports 8-bit video with 4:2:0 sampling. This profile is aimed at high-end consumer use and applications using high resolution but not needing the extended chroma formats or extended sample accuracy.
- High 10 profile – supports 4:2:0 with up to 10 bits of representation accuracy per sample
- High 4:2:2 profile – supports up to 4:2:2 chroma sampling along with up to 10 bits representation per sample.

- High 4:4:4 profile - supports up to 4:4:4 chroma sampling and up to 12 bits per sample. It supports the efficient lossless region coding and integer residual color transform. The integer residual color transform is used for RCT coding while avoiding color-space transformation error.

Table 3.1 Comparison of the high profiles of the FRExts [7]

Coding Tools	**High**	**High 10**	**High 4:2:2**	**High 4:4:4**
Main Profile Tools	X	X	X	X
4:2:0 Chroma Format	X	X	X	X
8 Bit Sample Bit Depth	X	X	X	X
8x8 vs. 4x4 Transform Adaptivity	X	X	X	X
Quantization Scaling Matrices	X	X	X	X
Separate Cb and Cr QP control	X	X	X	X
Monochrome video format	X	X	X	X
9 and 10 Bit Sample Bit Depth		X	X	X
4:2:2 Chroma Format			X	X
11 and 12 Bit Sample Bit Depth				X
4:4:4 Chroma Format				X
Residual Color Transform				X
Predictive Lossless Coding				X

The main application of the FRExts profiles is for more demanding high-fidelity applications. Hence the bit rate capabilities are increased for the FRExt profiles. Table 3.2 specifies the bit rate multiplier for the high profiles (the fourth column of table 2.1).

Table 3.2 Compressed bit rate multiplier for FRExts profiles [7]

FRExt Profile	Bit rate multiplier
High	1.25
High 10	3
High 4:2:2	4
High 4:4:4	4

3.4 YCgCo color space

Typically, a video is captured and displayed using the RGB (Red, Green and Blue) color space. The disadvantages of encoding the video in RGB domain are:

- Color components in the RGB domain are highly correlated.
- The response of the human visual system (HVS) is better matched to the luminance and chrominance components, rather than RGB. The HVS is very sensitive to the luminance information in the image. It is less sensitive to the chrominance components.

The YUV color space represents this luminance and chrominance information in a given RGB image. Hence the color conversion from the RGB domain to the YUV domain for encoding is performed. This conversion can be performed as follows [20]:

$$\begin{aligned} Y &= 0.299R + 0.587G + 0.114B \\ U &= -0.147R - 0.289G + 0.436B \\ V &= 0.615R - 0.515G - 0.100B \end{aligned} \tag{3.1}$$

This can be expressed in the matrix form as follows:

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.147 & -0.289 & 0.436 \\ 0.615 & -0.515 & -0.1 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{3.2}$$

In the YUV domain, the chrominance samples can be subsampled. This leads to compression. Then the inverse transform is performed from the YUV to RGB for display. YCbCr is a family of color spaces. Y stands for Luminance, Cb represents the

blue chroma and Cr represents the red chroma. The conversion from RGB to YCbCr can be performed as follows [7]:

$$Y = K_R * R + (1 - K_R - K_B) * G + K_B * B \tag{3.3}$$

$$C_b = ½ \left(\frac{B - Y}{1 - K_B} \right) \tag{3.4}$$

$$C_r = ½ \left(\frac{R - Y}{1 - K_R} \right) \tag{3.5}$$

with, e.g., $K_R = 0.2126$, $K_B = 0.0722$.

There are two problems with this approach:

- The samples are actually represented using integers. The rounding error is introduced in both the forward and inverse color transformations.
- The above transformation was not originally designed for digital video compression. It uses a sub-optimal trade-off between the complexity of the transformation (with difficult-to-implement coefficient values such as 0.2126 and 0.0722) and coding efficiency.

Considering the second problem, a new color space called YCgCo (where the "Cg" stands for green chroma and the "Co" stands for orange chroma) has been introduced. This is much simpler and typically has equal or better coding efficiency. The conversion from the RGB to the YCgCo color space can be performed as follows [7]:

$$Y = ½ \left[G + \left(\frac{R + B}{2} \right) \right] \tag{3.6}$$

$$Cg = ½ \left[G - \left(\frac{R + B}{2} \right) \right] \tag{3.7}$$

$$\text{Co} = \frac{(R - B)}{2} \tag{3.8}$$

This conversion reduces the complexity of conversion from the RGB domain to YCbCr and also increases the coding efficiency. The characteristics of the YCgCo color space can be explained as follows [8]:

- This color transform has been shown to be capable of achieving a decorrelation that is much better than that obtained by various RGB-to-YCbCr transforms and which, in fact, is very close to that of the Karhunen-Loeve transform [23].
- The transform is reversible in the sense that each original RGB triple can be exactly recovered from the corresponding YCgCo triple if the color difference components Co and Cg are represented with one additional bit accuracy relative to the bit depth used for representing RGB, and if furthermore, no information loss in any subsequent coding step is assumed.
- Both the forward and inverse RGB-to-YCgCo transforms require only a few shift and add operations per triple which, in addition, can be performed without the need of some extra memory apart from one single auxiliary register:

To obtain YCgCo components from RGB components, equations (3.9) are used.

$$\begin{aligned} Co &= R - B \\ t &= B + (Co >> 1) \\ Cg &= G - t \\ Y &= t + (Cg >> 1) \end{aligned} \tag{3.9}$$

To reconstruct the RGB components from the YCgCo components, equations (3.10) are used.

$$
\begin{aligned}
t &= Y - (Cg >> 1) \\
G &= Cg + t \\
B &= t - (Co >> 1) \\
R &= B + Co
\end{aligned}
\tag{3.10}
$$

The ">>"-operator denotes the bitwise right shift operator.

3.5 Principle of residual color transform

FRExts introduced residual color transform for 4:4:4 video. The input, output and the stored reference frames are retained in the RGB domain. The residual data is obtained after intra or inter prediction. This data is processed by the forward and inverse color transformations inside the encoder and decoder. This technique is called the residual color transform [7]. It eliminates the color space conversion error without significantly increasing the overall complexity of the system. The limitation of the RCT is that it can be applied only to 4:4:4 video. Its operation depends on the presence of luma and chroma components at all sample locations [7].

The residual color transform exploits the redundancy among the residual data of each RGB component after intra or inter prediction [9]. Even after performing inter/inter prediction, there is some correlation among the color components. To decorrelate this redundancy, the RGB components are transformed to YCgCo color space. The schematic of residual color transform can be shown in Fig. 3.7

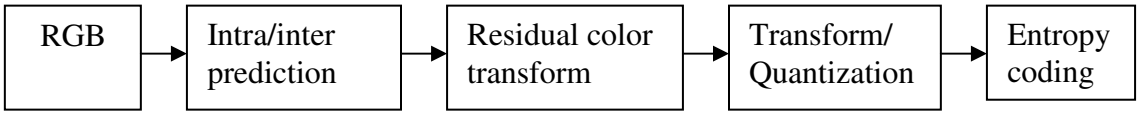

Fig. 3.7 Schematic of residual color transform [9]

3.6 Summary

This chapter gives the introduction to the fidelity range extension amendment of H.264. It also provides insight to the YCgCo color space and the residual color transform.

The following chapter explains the significance of lossless coding. It briefly explains the principle of arithmetic coding, followed by lossless coding implemented in the standards of JPEG-LS [30] and JPEG 2000 – lossless mode [29].

CHAPTER 4

LOSSLESS CODING

4.1 Introduction

Lossless image compression is aimed at representing an image signal with smallest possible number of bits without loss of any information. The lossy compression algorithms aim at achieving best possible fidelity within the available constraints of storage capacity or to represent an image in minimum possible number of bits to allow some possible loss of information. Lossless compression takes advantage of the redundancy present in the image signals, which is proportional to the amount of correlation among the image data samples [32]. Lossless coding demands that the decoded image should be identical both quantitatively and qualitatively to the original image. This requirement guarantees highest quality reconstructed output, but the compression achieved is less as compared to lossy coding. There are number of applications that demand high quality video. These applications led to the development of various lossless coding algorithms and lossless coding standards.

The following sections in this chapter explain the arithmetic coding algorithm. Due to it numerous advantages, this algorithm is selected in the proposed implementation. This chapter also briefly explains the lossless coding standards of JPEG 2000 [29] and JPEG-LS [30].

4.2 Arithmetic coding

Arithmetic coding is a very popular method of generating variable length codes [21]. This technique assigns codewords to particular sequences without having to generate codes for all sequences of that length. In order to understand this algorithm, the encoding process can be explained in following two parts. In first part, a unique identifier is generated for the sequence to be encoded. In the second part, a binary code is given to the tag. Hence, without generating codewords for all sequences, a sequence of length m can be encoded uniquely using arithmetic coding algorithm [21].

Arithmetic coding is generally suitable for encoding sequences with highly skewed probabilities. Use of Huffman coding [21] for encoding such sequences results in huge redundancy. The average code length exceeds entropy by a significant amount. Arithmetic coding overcomes this limitation of Huffman coding.

The implementation of arithmetic coding can be explained as follows. As explained, a unique identifier needs to be generated for a given sequence of symbols. The number of numbers in the unit interval, [0,1) is infinite. Hence it is possible to generate a distinct tag for a given sequence. The cumulative distribution function (cdf) [21] of a random variable will map the random variables in unit interval. If *X* is a random variable which has i number of symbols { a1, a2, a3, … , ai }, then probability density function of the random variable X is given as,

$$P(X = i) = P(a_i) \tag{4.1}$$

The cumulative distribution function of the random variable X is given as,

$$F_X(i) = \sum_{k=1}^{i} P(X = k) \tag{4.2}$$

The minimum value of *cdf* is 0 and the maximum value of *cdf* is 1. The unit interval is divided into subintervals of the form $[F_X(i-1), F_X(i)), i = 1,...,m$. The interval $[F_X(i-1), F_X(i))$ is associated with symbol a_i. The first symbol in the sequence restricts the interval containing the tags to one of these subintervals [21]. Let the first symbol in the sequence be a_k. Hence, the interval containing the tag value is the subinterval $[F_X(k-1), F_X(k))$. This subinterval is then partitioned in the same proportions as the original interval. Each succeeding symbol causes the tag to be restricted to a subinterval and this subinterval is then partitioned in the same proportions. The lower and the upper limits of the subinterval obtained after the last symbol are determined. The average of these limits is calculated and this average value is the tag of the given sequence. This tag is then converted to binary and transmitted over the network. There is variation of arithmetic coding where tags can be generated with scaling [21].

4.3 Lossless coding standards

4.3.1 Lossless JPEG standard

ITU and ISO/IEC jointly developed the JPEG standard for lossy and lossless compression of continuous tone, color or gray-scale, still images [32]. The following section briefly describes the lossless mode of the JPEG standard; it is also referred as lossless JPEG. The general coding structure of lossless JPEG can be explained in the following steps [32]:

- Prediction residuals are obtained by linear prediction/differential coding (DPCM). The entropy of these residuals is less than that of the original image. This leads to obtaining compression of the input image.
- The prediction is mapped to a symbol pair (*category, magnitude*). The symbol *category* signifies the number of bits required to encode the symbol *magnitude*.
- Huffman coding [21] is used to encode the *category* among pair of symbols (*category, magnitude*). The symbol *magnitude* is encoded using binary codeword whose length is given by symbol *category.* Arithmetic coding can be used instead of Huffman coding.

4.3.2 JPEG2000 standard

JPEG2000 is the latest standard for coding still images [32]. It was developed by Joint Photographic Experts Group (JPEG). It supports a number of new features over the JPEG standard. Some of the features are listed as follows [32]:

- Highly scalable code-streams with different progression orders (quality, resolution, spatial location and component)
- Lossy and lossless representations embedded with the same code-stream
- Region-of-interest (ROI) coding
- Support for continuous-tone, bi-level and compound image coding

JPEG2000 is divided into 12 different parts to address different application areas [32]. JPEG2000 Part 1 is the baseline standard. It describes the minimal code stream syntax to be followed for the compliance with the standard. All the other parts include the features supported by the baseline coding structure.

JPEG2000 implements wavelet-based bitplane coding [32]. If needed, the original image is divided into tiles, which are then coded independently. In order to decorrelate color images, two optional color transforms are defined in the standard. These transforms are irreversible color transform (ICT) and reversible color transform. These transform help increase compression efficiency. For lossless compression, reversible color transform is implemented. This transform can be implemented using finite precision arithmetic and is perfectly invertible [32]. The color image components are first divided into tiles and are coded separately. For each tile, if the image samples are unsigned pixel values, then the samples are first shifted in level so that they form a symmetric distribution of the discrete wavelet transform coefficients for low-low sub-band [32]. Two types of wavelet transforms are supported:

1. irreversible floating point 9/7 DWT [32]
2. reversible integer 5/3 DWT [34]

For lossless compression, the 5/3 DWT is used [32]. The transformed coefficients are quantized using a deadzone scalar quantizer for lossy coding scheme [35]. For lossless compression, quantization stage is bypassed. The coefficients in each subband are divided into coding blocks, usually of the size 64x64 or 32x32 [32]. Each of these blocks is independently bitplane coded from the most significant bit (MSB) to the least significant bit (LSB). The embedded block coding with optimal truncation (EBCOT) algorithm is used [32].

4.3.3 JPEG-LS standard

JPEG-LS standard is based on the LOw COmplexity LOssless COmpression for Images (LOCO-I) algorithm [30]. This standard is used to obtain lossless and near-lossless compression for continuous tone images. The block diagram of the JPEG-LS is shown in Fig. 4.1.

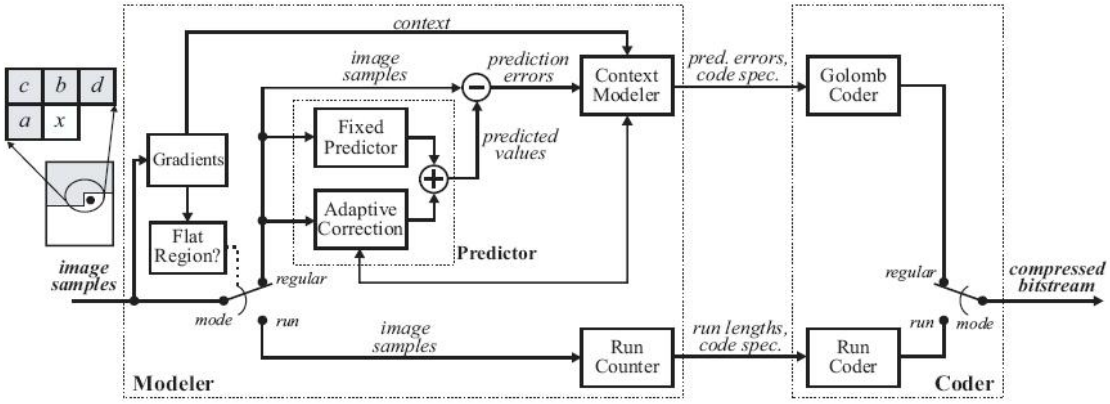

Fig. 4.1 Basic block diagram of JPEG-LS [30]

4.4 Summary

This chapter briefly describes the algorithm of arithmetic coding. It also explains the standards of lossless JPEG, JPEG2000 and JPEG-LS.

The next chapter explains the implementation of proposed algorithm and gives the results obtained from the proposed method and the JM software simulations.

CHAPTER 5

IMPLEMENTATION AND RESULTS

5.1 Proposed algorithm and results

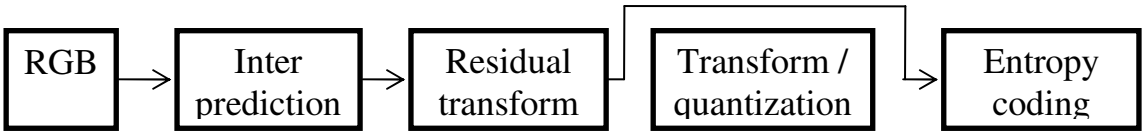

Fig. 5.1 Schematic of proposed algorithm – lossless coding

As the proposed algorithm is for lossless coding, the stage of transform and quantization is bypassed. The encoding and the decoding process in the proposed algorithm are illustrated in Fig. 5.2 and Fig. 5.3. Fig. 5.4 illustrates the transformation of the input sequence through the various color spaces.

The input to the algorithm is a video sequence in the 4:4:4 format. The previous frame in the sequence is considered as a reference frame for inter-prediction. The first frame in the sequence does not undergo any prediction. It is converted from RGB domain to YCgCo domain. The YCgCo coefficients are then encoded using arithmetic coding. Thus, the bitstream is obtained for the first frame. In the sequence, from the second frame onwards, the previous frame is the reference frame and the current frame is the frame to be predicted. Inter prediction is performed by subtracting the current frame from the previous frame (no motion compensation). The residual frame is obtained. This residual frame is in the 4:4:4 RGB domain. The residual frame is then transformed from RGB to YCgCo domain. These coefficients are then encoded using arithmetic coding.

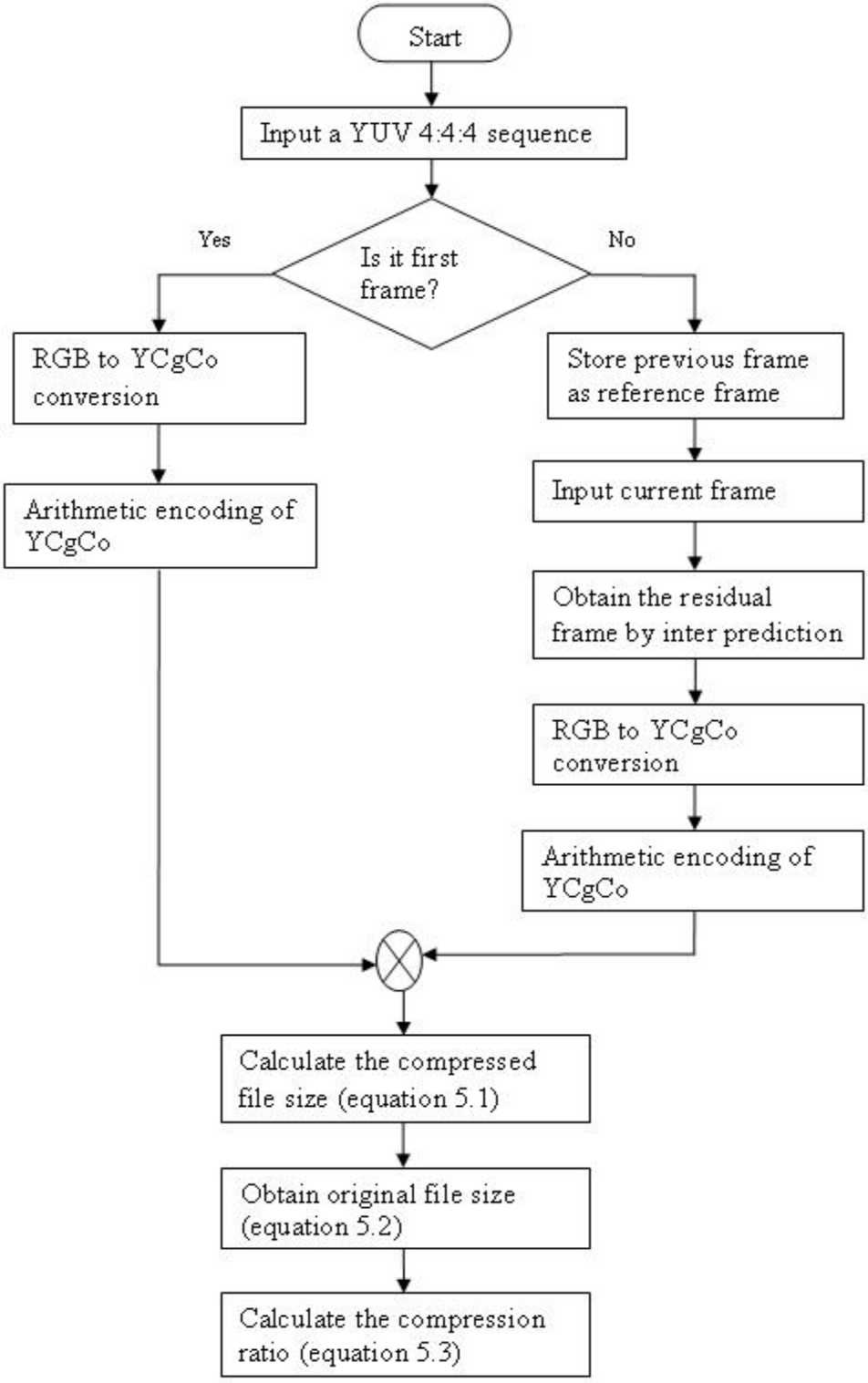

Fig. 5.2 Flowchart for encoding process in proposed algorithm

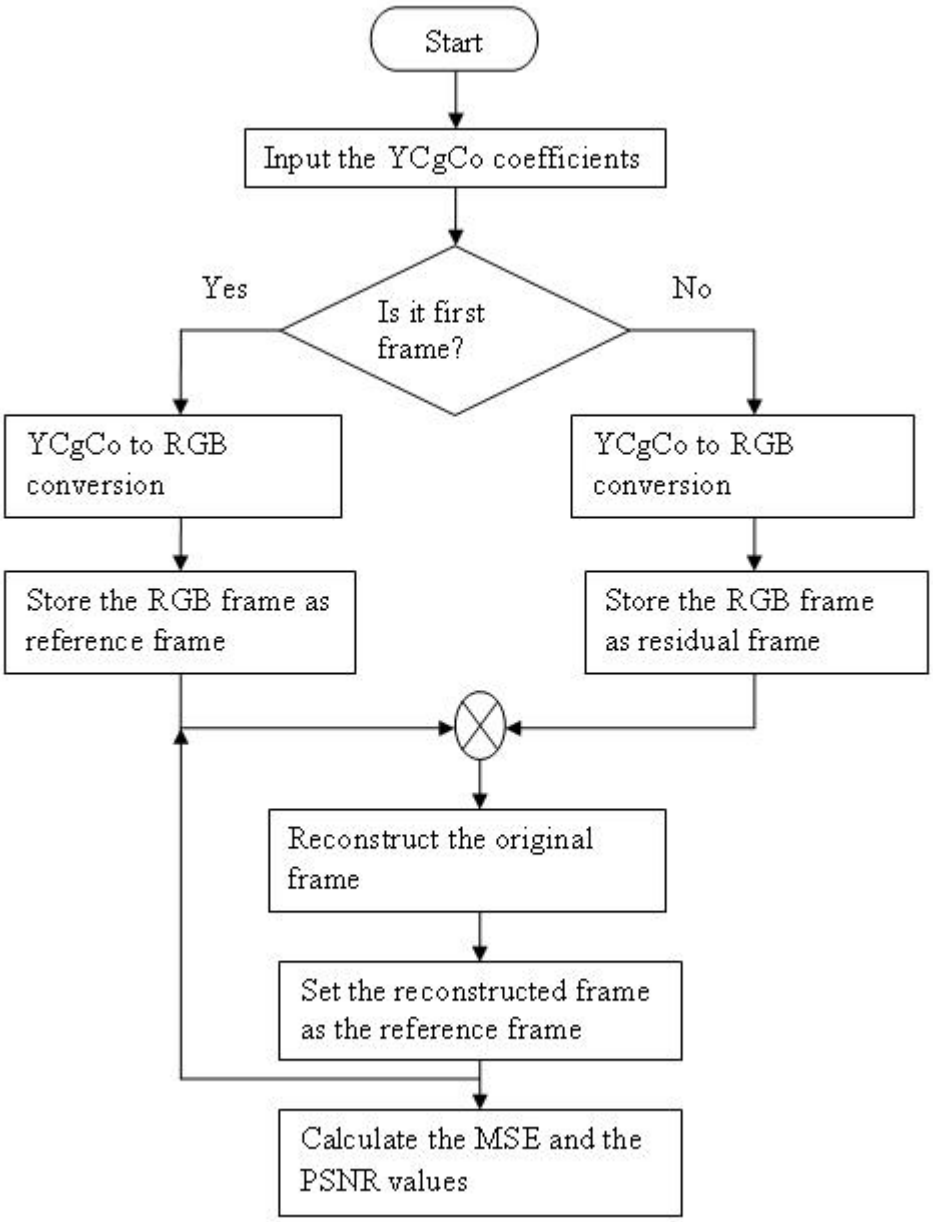

Fig. 5.3 Flowchart for the decoding process in proposed algorithm

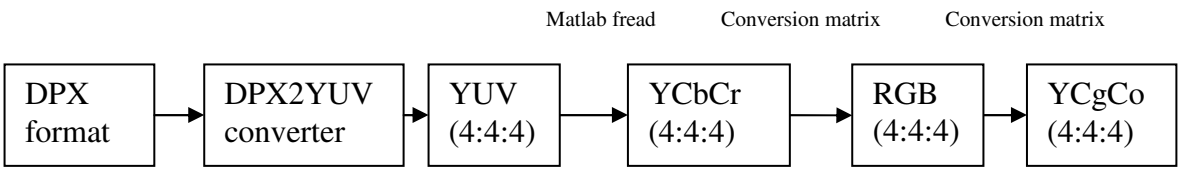

Fig. 5.4: Change in the color spaces during encoding (Refer to appendix B for details)

The size of all bit streams is aggregated and the size of the compressed file is obtained. This can be represented as using the following equation:

reconstructed_size = bitcount_Y + bitcount_Cg + bitcount_Co *(5.1)*

The size of the original sequence is obtained by using the following formula:

*original_size = row*column*3*8*number_of_frame (bits)* *(5.2)*

Hence, the compression ratio is calculated using the following formula:

$$compression\_ratio = \frac{original\_size}{reconstructed\_size} \tag{5.3}$$

The output sequence is then reconstructed from the YCgCo residuals and the reference frame.

The quality of the reconstructed image is the measured by computing the mean square error and the peak-signal-to-noise rate as follows:

$$MSE = \frac{\sum_{p=1}^{3}\sum_{m=1}^{columns}\sum_{n=1}^{rows}\left[original\_image(n,m,p) - reconstructed\_image(n,m,p)\right]^2}{rows * columns * 3} \tag{5.4}$$

$$PSNR(dB) = 20 * \log_{10}\left(\frac{255}{\sqrt{MSE}}\right) \tag{5.5}$$

Table 5.1 List of YUV test sequences

YUV Sequence	Resolution	Format	Number of frames
Waves.yuv	1920x1080	YUV 4:4:4	9
Night.yuv	1920x1080	YUV 4:4:4	9
Capitol.yuv	1920x1080	YUV 4:4:4	9
Freeway.yuv	1920x1080	YUV 4:4:4	9
Staples.yuv	1920x1080	YUV 4:4:4	9

Fig. 5.5 illustrates the original frame for the waves.yuv sequence. Fig. 5.6 illustrates the reconstructed frame for the waves.yuv sequence. Similarly, Fig. 5.7 and 5.8 illustrate the original frame and reconstructed frame for the sequence, freeway.yuv.

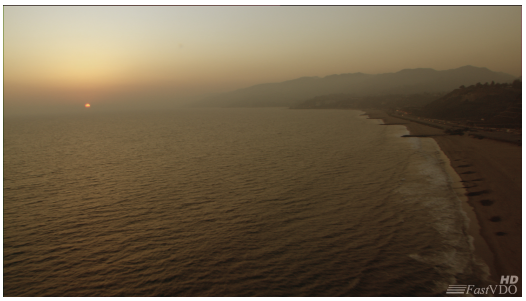

Fig. 5.5 Original frame – waves.yuv

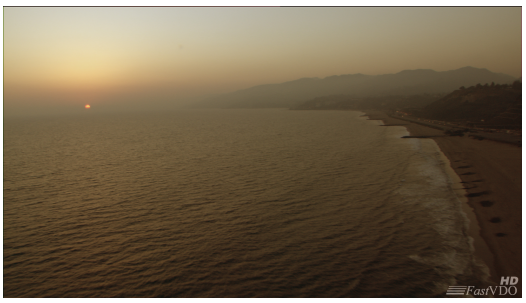

Fig. 5.6 Decoded frame - waves.yuv

Fig. 5.7 Original frame – freeway.yuv

Fig. 5.8 Decoded frame – freeway.yuv

Table 5.2: PSNR (dB) and MSE values for the various input sequences

Frame no.	waves.yuv	freeway.yuv	night.yuv	capitol.yuv	staples.yuv
1	328.9788	327.4766	342.9191	343.6534	342.5166
2	328.9602	327.4241	341.1691	341.7986	341.341
3	328.9398	327.3795	339.6881	340.2823	340.4543
4	328.9058	327.3333	338.5713	339.2134	339.7178
5	328.9058	327.2881	337.6628	338.278	339.2549
6	328.8922	327.2386	336.9104	337.5767	338.8337
MSE (in range of)	8.23E-29	1.16E-28	3.32E-30	2.80E-30	3.64E-30

Table 5.3: Original file size, compressed file size and compression ratio for various input test sequences

YUV Sequence	Original file size	Compressed bit stream file size	Compression ratio
Waves.yuv	298598400	138865894	2.1503
Night.yuv	298598400	85208503	3.5043
Capitol.yuv	298598400	86759986	3.4417
Freeway.yuv	298598400	130235548	2.2928
Staples.yuv	298598400	124387423	2.4006

5.2 Results from JM software simulation

The input sequences were encoded in H.264 using the latest version of the JM reference software 14.0 [14]. The encoder configurations selected for encoding the YUV 4:4:4 high definition sequences is explained as follows. The source width and height are set to 1920 and 1080, respectively. Nine frames of the video sequence are encoded. The profile selected for encoding YUV 4:4:4 frames in the JM reference software is High 4:4:4 profile defined in the FRExts. The level is set to 40. The level 40 corresponds to level 4 (see table 2.1 for details). This level supports high definition format at a frame rate of 60p/30i. The maximum compressed bit rate in non-FRExts

profiles, supported is 20Mbps. Maximum number of reference frames for typical picture size is 4. The entropy coding used is CABAC. Number of B frames to be encoded is set to zero. Lossless coding is achieved by setting the QPPrimeYZeroTransformBypassFlag to 1.

Figs. 5.9 through 5.14 illustrates the input original frames and the decoded output frames from the JM reference software for the sequences, waves.yuv, night.yuv and freeway.yuv.

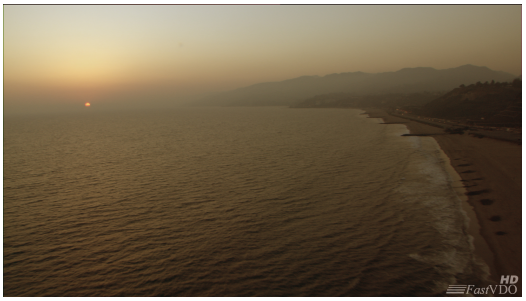

Fig. 5.9 Original frame – waves.yuv

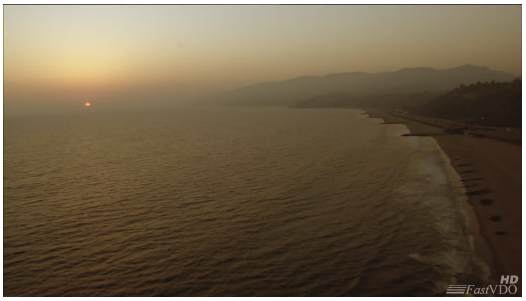

Fig. 5.10 Decoded frame – waves.yuv

Fig. 5.11 Original frame – night.yuv

Fig. 5.12 Decoded frame – night.yuv

Fig. 5.13 Original frame – freeway.yuv

Fig. 5.14 Decoded frame – freeway.yuv

Tables 5.4 through 5.8 list the SNR values, in dB, for every frame, obtained from the JM software simulations for the various input sequences.

Table 5.4 SNR values in dB for the 9 frames of sequence waves.yuv

Frame	Type	SNRY (dB)	SNRU (dB)	SNRV (dB)
1	IDR	70.065	69.277	69.214
2	P	69.589	68.942	68.853
3	P	69.992	69.231	69.197
4	P	69.914	69.217	69.145
5	P	69.913	69.174	69.119
6	P	69.930	69.155	69.162
7	P	69.919	69.236	69.125
8	P	69.905	69.172	69.086
9	P	69.957	69.182	69.141

Table 5.5 SNR values in dB for the 9 frames of sequence staples.yuv

Frame	Type	SNRY (dB)	SNRU (dB)	SNRV (dB)
1	IDR	72.377	72.879	72.340
2	P	71.787	72.261	71.730
3	P	71.699	72.304	71.847
4	P	71.776	72.315	71.995
5	P	71.729	72.287	71.880
6	P	71.868	72.397	71.943
7	P	71.757	72.248	71.814
8	P	71.809	72.390	71.932
9	P	71.774	72.289	71.820

Table 5.6 SNR values in dB for the 9 frames of sequence night.yuv

Frame	Type	SNRY (dB)	SNRU (dB)	SNRV (dB)
1	IDR	72.280	71.437	71.610
2	P	71.743	70.846	70.708
3	P	71.875	71.076	71.134
4	P	71.890	71.174	71.288
5	P	72.148	71.342	71.436
6	P	72.012	71.324	71.387
7	P	71.997	71.341	71.422
8	P	72.084	71.382	71.569
9	P	71.918	71.315	71.422

Table 5.7 SNR values in dB for the 9 frames of the sequence freeway.yuv

Frame	Type	SNRY (dB)	SNRU (dB)	SNRV (dB)
1	IDR	70.942	70.323	71.070
2	P	66.868	66.845	67.081
3	P	67.083	67.077	67.376
4	P	67.132	67.037	67.375
5	P	67.195	67.083	67.324
6	P	67.087	67.081	67.354
7	P	67.131	67.086	67.358
8	P	67.172	61.106	67.393
9	P	67.167	67.064	67.304

Table 5.8 SNR values in dB for the 9 frames of the sequence capitol.yuv

Frame	Type	SNRY (dB)	SNRU (dB)	SNRV (dB)
1	IDR	72.587	71.641	71.773
2	P	71.809	70.814	70.584
3	P	72.023	71.258	71.349
4	P	72.322	71.479	71.514
5	P	72.219	71.475	71.504
6	P	72.299	71.498	71.557
7	P	72.162	71.508	71.644
8	P	72.150	71.514	71.652
9	P	72.393	71.759	71.929

Table 5.9 Original file size, compressed file size and compression ratio from JM software simulations

YUV Sequence	Original file size (bytes)	Compressed file size (bytes)	Compression ratio
Waves.yuv	55,987,200	18,606,996	3.009
Night.yuv	55,987,200	11,109,828	5.039
Capitol.yuv	55,987,200	11,558,912	4.844
Freeway.yuv	55,987,200	19,997,793	2.799
Staples.yuv	55,987,200	19,776,431	2.831

5.3 Results from JPEG software simulations

5.3.1: Independent JPEG:

The test sequences were encoded using the independent JPEG group's JPEG software [28]. This software does not give lossless compression. This accounts for the reasonable PSNR values. Fairly high compression ratios have been obtained for nearly lossless compression.

As JPEG compression standard is for coding of still images, the yuv video sequence is processed on a frame-by-frame basis. A Matlab code (see appendix A) have been written to obtain portable pixel map (.ppm) images from the YUV video sequence. Tables 5.10 through 5.14 list out the compression ratios obtained and the PSNR values from the JPEG software simulations for the five test sequences. The file size is measured in bytes.

Table 5.10: Compression ratio and PSNR for waves.yuv for independent JPEG simulation

Frame no.	Original file size	Compressed file size	Reconstructed file size	Compression ratio	PSNR value (dB)
1	6220817	1217332	6220817	5.110205761	46.2812
2	6220817	1222474	6220817	5.088711089	46.2323
3	6220817	1224968	6220817	5.078350618	46.2334
4	6220817	1228699	6220817	5.062929977	46.2064
5	6220817	1221448	6220817	5.092985538	46.316
6	6220817	1216739	6220817	5.112696314	46.3789
7	6220817	1225256	6220817	5.077156937	46.2861
8	6220817	1226592	6220817	5.071626914	46.291
9	6220817	1223653	6220817	5.083808073	46.3259
			AVERAGE	**5.086496802**	**46.2835**

Table 5.11 Compression ratio and PSNR for freeway.yuv for independent JPEG simulation

Frame no.	Original file size	Compressed file size	Reconstructed file size	Compression ratio	PSNR value (dB)
1	6220817	1579519	6220817	3.938424926	40.4234
2	6220817	1580305	6220817	3.936466062	40.411
3	6220817	1580485	6220817	3.936017741	40.417
4	6220817	1580410	6220817	3.936204529	40.4354
5	6220817	1581235	6220817	3.934150838	40.4385
6	6220817	1582170	6220817	3.93182591	40.4475
7	6220817	1582152	6220817	3.931870642	40.4513
8	6220817	1581309	6220817	3.933966733	40.4889
9	6220817	1581646	6220817	3.933128526	40.4947
			AVERAGE	**3.934672878**	**40.4453**

Table 5.12 Compression ratio and PSNR for night.yuv for independent JPEG simulation

Frame no.	Original file size	Compressed file size	Reconstructed file size	Compression ratio	PSNR value (dB)
1	6220817	761530	6220817	8.168840361	44.1449
2	6220817	765053	6220817	8.131223588	44.1817
3	6220817	757918	6220817	8.207770498	44.1978
4	6220817	761249	6220817	8.171855727	44.1802
5	6220817	755276	6220817	8.236481763	44.282
6	6220817	763999	6220817	8.142441286	44.3092
7	6220817	762017	6220817	8.163619709	44.5696
8	6220817	752031	6220817	8.272022031	44.6579
9	6220817	759138	6220817	8.194579905	44.4279
			AVERAGE	**8.187648319**	**44.3279**

Table 5.13 Compression ratio and PSNR for capitol.yuv for independent JPEG simulation

Frame no.	Original file size	Compressed file size	Reconstructed file size	Compression ratio	PSNR value (dB)
1	6220817	803954	6220817	7.737777286	43.8418
2	6220817	803769	6220817	7.739558256	43.676
3	6220817	807070	6220817	7.70790266	43.636
4	6220817	806804	6220817	7.710443924	43.6119
5	6220817	810879	6220817	7.671695777	43.651
6	6220817	813992	6220817	7.642356436	43.5799
7	6220817	814246	6220817	7.639972441	43.5621
8	6220817	824330	6220817	7.546512926	43.4759
9	6220817	814630	6220817	7.636371113	43.5209
			AVERAGE	**7.670287869**	**43.6173**

Table 5.14 Compression ratio and PSNR value for staples.yuv for independent JPEG simulation

Frame no.	Original file size	Compressed file size	Reconstructed file size	Compression ratio	PSNR value (dB)
1	6220817	1288837	6220817	4.826690264	41.0765
2	6220817	1289465	6220817	4.824339552	41.1195
3	6220817	1291584	6220817	4.816424638	41.2066
4	6220817	1296960	6220817	4.796460184	41.1756
5	6220817	1299409	6220817	4.787420281	41.0731
6	6220817	1294205	6220817	4.806670504	41.1459
7	6220817	1292626	6220817	4.812542066	41.0565
8	6220817	1294032	6220817	4.807313111	41.1587
9	6220817	1297150	6220817	4.795757622	41.0692
			AVERAGE	**4.808179803**	**41.1202**

5.3.2: JPEG-2000

The test sequences were encoded in the JPEG-2000 format using the JasPer software [29]. The test sequences were processed on a frame-by-frame basis. The YUV frames were converted to portable pixel map using the Matlab code (see appendix A).

Tables 5.15 through 5.19 list the compression ratio and the PSNR values for the five test sequences. The JPEG 2000 ensures lossless coding. Hence, the PSNR values for most of the frames was infinity (Inf.). Some of the frames had very high PSNR values in the range of 106 dB and above. The file size is measured in bytes.

Table 5.15 Compression ratio and PSNR values for waves.yuv for JPEG-2000 simulation

Frame no.	Original file size	Compressed file size	Reconstructed file size	Compression ratio	PSNR value (dB)
1	6220817	2297096	6220817	2.708122342	108.2878
2	6220817	2302603	6220817	2.701645486	113.059
3	6220817	2306894	6220817	2.696620217	111.2981
4	6220817	2311097	6220817	2.691716098	107.6183
5	6220817	2301455	6220817	2.702993107	98.9936
6	6220817	2295800	6220817	2.709651102	102.0899
7	6220817	2305857	6220817	2.697832953	99.6347
8	6220817	2307869	6220817	2.695480983	111.2981
9	6220817	2302151	6220817	2.702175922	104.3084
			AVERAGE	**2.700693134**	**106.2875**

Table 5.16 Compression ratio and PSNR values for freeway.yuv for JPEG-2000 simulation

Frame no.	Original file size	Compressed file size	Reconstructed file size	Compression ratio	PSNR value (dB)
1	6220817	2855874	6220817	2.178253312	Inf.
2	6220817	2856672	6220817	2.177644826	Inf.
3	6220817	2856476	6220817	2.177794247	Inf.
4	6220817	2856570	6220817	2.177722583	113.059
5	6220817	2857496	6220817	2.177016871	113.059
6	6220817	2858373	6220817	2.176348923	Inf
7	6220817	2858525	6220817	2.176233197	116.0693
8	6220817	2857401	6220817	2.17708925	Inf
9	6220817	2858611	6220817	2.176167726	111.2981
			AVERAGE	**2.177141215**	**Inf.**

Table 5.17 Compression ratio and PSNR values for capitol.yuv for JPEG-2000 simulation

Frame no.	Original file size	Compressed file size	Reconstructed file size	Compression ratio	PSNR value (dB)
1	6220817	1500885	6220817	4.144765921	Inf.
2	6220817	1504415	6220817	4.135040531	Inf.
3	6220817	1510920	6220817	4.117237842	Inf.
4	6220817	1507483	6220817	4.126624977	Inf.
5	6220817	1517502	6220817	4.09937977	108.2878
6	6220817	1523747	6220817	4.08257867	107.6183
7	6220817	1516769	6220817	4.101360853	Inf.
8	6220817	1537929	6220817	4.044931203	111.2981
9	6220817	1524849	6220817	4.079628212	Inf.
			AVERAGE	**4.103505331**	**Inf.**

Table 5.18 Compression ratio and PSNR value for night.yuv for JPEG-2000 simulation

Frame no.	Original file size	Compressed file size	Reconstructed file size	Compression ratio	PSNR value (dB)
1	6220817	1459461	6220817	4.262407149	Inf.
2	6220817	1460037	6220817	4.260725584	107.6183
3	6220817	1445637	6220817	4.303166701	108.2878
4	6220817	1457822	6220817	4.267199288	113.059
5	6220817	1442337	6220817	4.313012146	113.059
6	6220817	1458285	6220817	4.265844468	110.0487
7	6220817	1458041	6220817	4.266558348	101.4453
8	6220817	1439630	6220817	4.321122094	104.0281
9	6220817	1452053	6220817	4.284152851	Inf.
			AVERAGE	**4.282687625**	**Inf.**

Table 5.19 Compression ratio and PSNR value for staples.yuv for JPEG-2000 simulation

Frame no.	Original file size	Compressed file size	Reconstructed file size	Compression ratio	PSNR value (dB)
1	6220817	2465999	6220817	2.522635654	Inf.
2	6220817	2469080	6220817	2.519487825	96.5268
3	6220817	2474521	6220817	2.513947952	110.0487
4	6220817	2480883	6220817	2.50750116	100.3872
5	6220817	2483778	6220817	2.504578509	93.1467
6	6220817	2474879	6220817	2.5135843	97.0384
7	6220817	2472470	6220817	2.516033359	100.8841
8	6220817	2478057	6220817	2.510360738	98.2878
9	6220817	2479503	6220817	2.508896743	97.4959
			AVERAGE	**2.513002916**	**Inf.**

5.3.3 JPEG-LS

The test sequences were encoded in the JPEG-LS standard using the JPEG-LS software obtained from [50]. The compression ratio and the PSNR value obtained from encoding the test sequences in the JPEG-LS format is listed in tables 5.20 through 5.24. The file size is measured in bytes.

Table 5.20 Compression ratio and PSNR value for waves.yuv for JPEG-LS simulation

Frame no.	Original file size	Compressed file size	Reconstructed file size	Compression ratio	PSNR value (dB)
1	6220817	2168526	6220817	2.868684535	46.2812
2	6220817	2175180	6220817	2.859909065	46.2323
3	6220817	2177683	6220817	2.856621923	46.2334
4	6220817	2183197	6220817	2.849407085	46.2064
5	6220817	2173942	6220817	2.861537704	46.316
6	6220817	2165538	6220817	2.872642734	46.3789
7	6220817	2178405	6220817	2.855675138	46.2861
8	6220817	2180395	6220817	2.853068825	46.291
9	6220817	2172201	6220817	2.863831202	46.3259
			AVERAGE	**2.860153135**	**46.2835**

Table 5.21 Compression ratio and PSNR value for freeway.yuv for JPEG-LS simulation

Frame no.	Original file size	Compressed file size	Reconstructed file size	Compression ratio	PSNR value (dB)
1	6220817	2749607	6220817	2.262438596	40.4234
2	6220817	2750733	6220817	2.261512477	40.411
3	6220817	2750604	6220817	2.261618539	40.417
4	6220817	2750618	6220817	2.261607028	40.4354
5	6220817	2751834	6220817	2.260607653	40.4385
6	6220817	2752311	6220817	2.26021587	40.4475
7	6220817	2751483	6220817	2.260896033	40.4513
8	6220817	2751488	6220817	2.260891925	40.4889
9	6220817	2751852	6220817	2.260592866	40.4947
			AVERAGE	**2.261153443**	**40.4453**

Table 5.22 Compression ratio and PSNR value for night.yuv for JPEG-LS simulation

Frame no.	Original file size	Compressed file size	Reconstructed file size	Compression ratio	PSNR value (dB)
1	6220817	1155525	6220817	5.38354168	44.1449
2	6220817	1159045	6220817	5.367191955	44.1817
3	6220817	1148139	6220817	5.418174106	44.1978
4	6220817	1158830	6220817	5.368187741	44.1802
5	6220817	1140529	6220817	5.454326019	44.282
6	6220817	1155927	6220817	5.381669431	44.3092
7	6220817	1155656	6220817	5.382931426	44.5696
8	6220817	1138416	6220817	5.464449727	44.6579
9	6220817	1149187	6220817	5.413233007	44.4279
			AVERAGE	**5.40374501**	**44.3279**

Table 5.23 Compression ratio and PSNR value for capitol.yuv for JPEG-LS simulation

Frame no.	Original file size	Compressed file size	Reconstructed file size	Compression ratio	PSNR value (dB)
1	6220817	1201526	6220817	5.177430201	43.8418
2	6220817	1204825	6220817	5.163253585	43.676
3	6220817	1211200	6220817	5.136077444	43.636
4	6220817	1207140	6220817	5.153351724	43.6119
5	6220817	1217167	6220817	5.110898504	43.651
6	6220817	1221234	6220817	5.093877996	43.5799
7	6220817	1217629	6220817	5.108959297	43.5621
8	6220817	1235230	6220817	5.036160877	43.4759
9	6220817	1222902	6220817	5.086930106	43.5209
			AVERAGE	**5.118548859**	**43.6173**

Table 5.24 Compression ratio and PSNR value for staples.yuv for JPEG-LS simulation

Frame no.	Original file size	Compressed file size	Reconstructed file size	Compression ratio	PSNR value (dB)
1	6220817	2083801	6220817	2.985322015	41.0765
2	6220817	2089288	6220817	2.977481802	41.1195
3	6220817	2091208	6220817	2.974748088	41.2066
4	6220817	2097758	6220817	2.965459791	41.1756
5	6220817	2101665	6220817	2.959946994	41.0731
6	6220817	2092690	6220817	2.972641433	41.1459
7	6220817	2088560	6220817	2.97851965	41.0565
8	6220817	2094360	6220817	2.970271109	41.1587
9	6220817	2096235	6220817	2.967614318	41.0692
			AVERAGE	**2.972445022**	**41.1202**

5.4 Summary

This chapter explains the implementation of the proposed algorithm. It also describes the results obtained and the input and output images obtained from the proposed method. This chapter further explains the results, input and output images obtained by encoding and decoding the test sequences using the JM reference software.

The following chapter explains the conclusions of this research and suggests possible future work.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

The proposed algorithm, implementing the concepts of YCgCo color space and residual color transform to achieve lossless coding, was successfully implemented to obtain lossless compression for high definition YUV sequences. The sampling format of these sequences is 4:4:4. The reconstructed image at the output was of high quality as demonstrated by the high PSNR value. The test sequences are successfully encoded in H.264/MPEG-4 AVC standard using the JM reference software.

6.2 Future work

The coding efficiency of the proposed algorithm can be further improved by implementing block-based coding. This would reduce the time taken for encoding the YUV sequences and will make the code more memory efficient. The YUV frames do not have any header information. By implementing some header information, this algorithm can be used to implement presentation time stamps. This can be used for achieving audio-video synchronization.

APPENDIX A

MATLAB SOURCE CODE

The following source code implements the proposed algorithm in the MATLAB platform. The input to this code is a 4:4:4 YUV sequence. The YUV frames are converted in RGB domain for encoding. Inter prediction is performed by considering the previous frame as a reference frame. The residual frame is converted into YCgCo domain. These YCgCo coefficients are coded using arithmetic coding. The size of the bitstream is calculated. Compression ratio is calculated by comparing the original size of the raw image and the compressed bit stream.

File 1: residual_ycgco_4.m

```
close all; clear all; clc;
%read the input file
%inputsequence='plane.yuv';
inputsequence=input('Enter the file name to be processed = ', 's');
%enter the number of frames
totalFrames = input('Enter the number of frames to be processed in the sequence = ');
%display the value entered
disp('The input file is = ')
disp(inputsequence)
disp('The number of frames to be processed = ')
disp(totalFrames)
%initialize the values
samplingFormat=[1920 1080];
fid=fopen(inputsequence,'rb');
%read the YCbCr values from the input sequence.
for i=1:1:totalFrames
   [y] = fread(fid, [1920 1080], 'uint8');
   y = y';
   Y(:,:,i) = y;
```

```
    [cb] = fread(fid, [1920 1080], 'uint8');
    cb = cb';
    CB(:,:,i) = cb;
    [cr] = fread(fid, [1920 1080], 'uint8');
    cr = cr';
    CR(:,:,i) = cr;
end
fclose(fid);
clear samplingFormat;
clear y;
clear cb;
clear cr;
clear inputsequence;
%clear fid;
%convert the YCbCr to RGB sequence and display the original frames
for i=1:1:totalFrames
    red(:,:) = 1.164*(Y(:,:,i)-16) + 1.596*(CR(:,:,i)-128);
    green(:,:) = 1.164 * (Y(:,:,i)-16) - 0.813*(CR(:,:,i)-128) - 0.392*(CB(:,:,i)-128);
    blue(:,:) = 1.164 * (Y(:,:,i)-16) + 2.017*(CB(:,:,i)-128);
    image_rgb(:,:,1) = red(:,:);
    image_rgb(:,:,2) = green(:,:);
    image_rgb(:,:,3) = blue(:,:);
    rgb_sequence(:,:,:,i) = image_rgb(:,:,:);
    clear image_rgb;
    clear red;
    clear green;
    clear blue;
end
clear Y;
```

```
clear CB;
clear CR;
bitcount_Y = 0;
bitcount_Cg = 0;
bitcount_Co = 0;
%encode the first frame
first_frame_RGB (:,:,:) = rgb_sequence(:,:,:,1);
first_frame_YCGCO(:,:,:) = rgb2ycgco (first_frame_RGB (:,:,:) );
bitcount_Y = arcoder(first_frame_YCGCO (:,:,1) );
bitcount_Cg = arcoder(first_frame_YCGCO (:,:,2) );
bitcount_Co = arcoder ( first_frame_YCGCO (:,:,3) );
%clear first_frame_RGB;
%decode the first frame
recon_referenceRGB(:,:,:) = ycgco2rgb(first_frame_YCGCO (:,:,:) );
psnrfirstframe = metcaldouble(first_frame_RGB, recon_referenceRGB)
%encode the second frames onwards
for i=1:1:(totalFrames-1)
   reference_frame_RGB = rgb_sequence(:,:,:,i);
   j=i+1;
   current_frame_RGB = rgb_sequence(:,:,:,j);
   residual_frame_RGB = current_frame_RGB - reference_frame_RGB;
   figure
   imshow(uint8(current_frame_RGB(:,:,:)))
   title(['Original Frame RGB Number = ', num2str(j)])
    figure
    imshow(uint8(residual_frame_RGB(:,:,:)))
    title( ['Residual Frame RGB Number = ', num2str(j)] )
   residual_frame_YCGCO(:,:,:) = rgb2ycgco (residual_frame_RGB(:,:,:) );
%      figure
```

```
%      imshow(uint8(residual_frame_YCGCO(:,:,:)))
%      title( ['Residual Frame YCGCO Number = ', num2str(j)] )
    bitcount_Y = bitcount_Y + arcoder (residual_frame_YCGCO (:,:,1) );
    bitcount_Cg = bitcount_Cg + arcoder (residual_frame_YCGCO (:,:,2) );
    bitcount_Co = bitcount_Co + arcoder (residual_frame_YCGCO (:,:,3) );
    % decode the sequence
    recon_residual_frameRGB(:,:,:) = ycgco2rgb(residual_frame_YCGCO (:,:,:) );
    recon_current_frameRGB (:,:,:) = recon_referenceRGB(:,:,:) +
recon_residual_frameRGB(:,:,:);
    recon_referenceRGB(:,:,:) = recon_current_frameRGB(:,:,:);
    figure
    imshow(uint8(recon_referenceRGB(:,:,:) ));
    title(['Reconstructed Frame. Number = ',num2str(j)])
    errorRGBFrame(:,:,:) = 2*( current_frame_RGB(:,:,:) - recon_referenceRGB(:,:,:) ) +
128;
    figure
    imshow(uint8(errorRGBFrame(:,:,:)))
    title(['Error Frame Number = ', num2str(j)])
    %calculate the MSE and the PSNR values
    psnr_frame = metcal(current_frame_RGB,recon_current_frameRGB)
end
compressed_file_size = bitcount_Y + bitcount_Cg + bitcount_Co
original_file_size = 1920*1080*3*8*totalFrames
compression_ratio = original_file_size/compressed_file_size
```

File 2: rgb2ycgco.m

```
function [outputSequence] = rgb2ycgco(inputSequence)
%this function converts the input RGB sequence to YCGCO sequence
%calculate the RGB components of the residual frame
```

```
redResidual(:,:) = inputSequence(:,:,1);
greenResidual(:,:) = inputSequence(:,:,2);
blueResidual(:,:) = inputSequence(:,:,3);
clear inputSequence;
%convert the residuals from RGB to YCgCo
Y1new(:,:) = (greenResidual + (redResidual(:,:) + blueResidual(:,:))/2)/2;
Cgnew(:,:) = (greenResidual - (redResidual(:,:) + blueResidual(:,:))/2)/2;
Conew(:,:) = (redResidual(:,:) - blueResidual(:,:))/2;
clear redResidual;
clear blueResidual;
clear greenResidual;
clear t;
outputSequence(:,:,1) = Y1new(:,:);
outputSequence(:,:,2) = Cgnew(:,:);
outputSequence(:,:,3) = Conew(:,:);
```

File 3: ycgco2rgb.m

```
function [RGBOutputSequence] = ycgco2rgb (YCGCOInputSequence)
%this function converts YCGCO to RGB
Y1(:,:) = YCGCOInputSequence (:,:,1);
Cg(:,:) = YCGCOInputSequence (:,:,2);
Co(:,:) = YCGCOInputSequence (:,:,3);
red_recon = Y1 + Co - Cg;
green_recon = Y1 + Cg;
blue_recon = Y1 - Co - Cg;
RGBOutputSequence(:,:,1) = red_recon(:,:);
RGBOutputSequence(:,:,2) = green_recon(:,:);
RGBOutputSequence(:,:,3) = blue_recon(:,:);
```

File 4: arcoder.m

```
function code_length = arcoder (seq)
%input to this function is the sequence to be encoded
seq = round(seq);
seq_vector = seq(:)';
symb = [min(seq_vector):max(seq_vector)];
ncount=histc(seq_vector,symb);
a = ncount(ncount~=0);
b = symb(ncount~=0);
seq_tmp = seq_vector;
for j=1:length(b)
   seq_tmp(seq_vector == b(j)) = j;
end
code_length = length(arithenco(seq_tmp,a));
```

File 5: metcaldouble.m

```
function psnr = metcaldouble (originalimage, reconstructedimage)
%this function takes the original_image, reconstructed_image and returns
%the values of PSNR
[img_size] = size(originalimage);
row = img_size(1);
column = img_size(2);
MSE = 0;
temp = 0;
for p=1:1:3
   for i=1:1:row
      for j=1:1:column
         temp = double((originalimage(i,j,p) - reconstructedimage(i,j,p)));
         temp2 = temp^2;
```

```
            MSE = MSE + temp2;
        end
    end
end
MSE = MSE / (3*row*column)
max_pel = 255;
temp = 0;
temp = max_pel/sqrt(MSE);
psnr = 20 * log10(temp);
```

APPENDIX B

STEPS TO DOWNLOAD HIGH DEFINITION SEQUENCES

The 4:4:4 HD test sequences are available on the server - ftp.tnt.uni-hannover.de. To get these sequences, following steps can be implemented:

Step 1: Download ftp client from internet.

Step 2: Log in to the ftp.tnt.uni-hannover.de. Username and password is required.

Step 3: The test sequences are in the path: testsequences/Viper_testset_FastVDO.

Step 4: The test sequences are in dpx format. To convert them to YUV 4:4:4 format, download dpx2yuv converter from FastVDO website at - http://www.fastvdo.com/DPX2YUV.html.

The resolution of the video frame is 1920*1080 and the YUV format is 4:4:4.

APPENDIX C

ENCODER CONFIGURATION FILE USED FOR JM SOFTWARE SIMULATIONS

The encoder in the JM reference software is configured using the following configuration file to obtain lossless coding.

```
# New Input File Format is as follows
# <ParameterName> = <ParameterValue> # Comment
#
# See configfile.h for a list of supported ParameterNames
#
# For bug reporting and known issues see:
# https://ipbt.hhi.de


###############################################################################
# Files
###############################################################################
InputFile             = "capitol9Frames.yuv"       # Input sequence
InputHeaderLength     = 0      # If the inputfile has a header, state
it's length in byte here
StartFrame            = 0      # Start frame for encoding. (0-N)
FramesToBeEncoded     = 9      # Number of frames to be coded
FrameRate             = 30.0   # Frame Rate per second (0.1-100.0)
SourceWidth           = 1920   # Source frame width
SourceHeight          = 1080   # Source frame height
SourceResize          = 0      # Resize source size for output
OutputWidth           = 1920   # Output frame width
OutputHeight          = 1080   # Output frame height
TraceFile             = "capitoltrace_enc2.txt"      # Trace file
ReconFile             = "capitoltest_rec2.yuv"       # Recontruction
YUV file
OutputFile            = "capitoltest2.264"           # Bitstream
StatsFile             = "capitolstats2.dat"          # Coding
statistics file

###############################################################################
# Encoder Control
###############################################################################
ProfileIDC            = 244 # Profile IDC (66=baseline, 77=main,
88=extended; FREXT Profiles: 100=High, 110=High 10, 122=High 4:2:2,
244=High 4:4:4, 44=CAVLC 4:4:4 Intra)
IntraProfile          = 0   # Activate Intra Profile for FRExt (0:
false, 1: true)
                            # (e.g. ProfileIDC=110, IntraProfile=1  =>
High 10 Intra Profile)
LevelIDC              = 42  # Level IDC   (e.g. 20 = level 2.0)

IntraPeriod           = 0   # Period of I-pictures   (0=only first)
IDRPeriod             = 0   # Period of IDR pictures (0=only first)
AdaptiveIntraPeriod   = 1   # Adaptive intra period
AdaptiveIDRPeriod     = 1   # Adaptive IDR period
IntraDelay            = 0   # Intra (IDR) picture delay (i.e. coding
structure of PPIPPP... )
```

```
EnableIDRGOP          = 0   # Support for IDR closed GOPs (0:
disabled, 1: enabled)
EnableOpenGOP         = 0   # Support for open GOPs (0: disabled, 1:
enabled)
QPISlice              = 1  # Quant. param for I Slices (0-51)
QPPSlice              = 1  # Quant. param for P Slices (0-51)
FrameSkip             = 0   # Number of frames to be skipped in input
(e.g 2 will code every third frame)
ChromaQPOffset        = 0   # Chroma QP offset (-51..51)

DisableSubpelME       = 0   # Disable Subpixel Motion Estimation
(0=off/default, 1=on)
SearchRange           = 32  # Max search range


NumberReferenceFrames = 1   # Number of previous frames used for inter
motion search (0-16)

###########################################################################
# B Slices
###########################################################################

NumberBFrames          = 0  # Number of B coded frames inserted (0=not
used)

###########################################################################
#FREXT stuff
###########################################################################

YUVFormat               = 3     # YUV format (0=4:0:0, 1=4:2:0,
2=4:2:2, 3=4:4:4)
RGBInput                = 0     # 1=RGB input, 0=GBR or YUV input
SeparateColourPlane     = 0     # 4:4:4 coding: 0=Common mode,
1=Independent mode
SourceBitDepthLuma      = 8     # Source Bit Depth for Luma color
component (8...14 bits)
SourceBitDepthChroma    = 8     # Source Bit Depth for Chroma color
components (8...14 bits)
SourceBitDepthRescale   = 0     # Rescale bit depth of source for
output (0: Disable 1: Enable)
OutputBitDepthLuma      = 8     # Output Bit Depth for Luma color
component (8...14 bits)
OutputBitDepthChroma    = 8     # Output Bit Depth for Chroma color
components (8...14 bits)

CbQPOffset              = 0     # Chroma QP offset for Cb-part (-
51..51)
CrQPOffset              = 0     # Chroma QP offset for Cr-part (-
51..51)
Transform8x8Mode        = 1     # (0: only 4x4 transform, 1: allow
using 8x8 transform additionally, 2: only 8x8 transform)
ReportFrameStats        = 0     # (0:Disable Frame Statistics 1:
Enable)
```

```
DisplayEncParams        = 0     # (0:Disable Display of Encoder Params
1: Enable)
Verbose                 = 1     # level of display verboseness
(0:short, 1:normal, 2:detailed)

###########################################################################
#Lossless Coding (FREXT)
###########################################################################

QPPrimeYZeroTransformBypassFlag = 1    # Enable lossless coding when
qpprime_y is zero (0 Disabled, 1 Enabled)
```

APPENDIX D

KEY TECHNICAL AREA (KTA) SOFTWARE

1 Introduction

In July 2005, Video Coding Experts Group (VCEG) decoded to establish the KTA software, at the Busan meeting [39]. This effort was to gather various coding efficiency tools to continue progress ahead of H.264/MPEG-4 AVC standard. These coding tools are implemented in the KTA software. The latest version of the KTA software is available at [40]. This version is based on JM reference software version 11.

2 Residual color transform in KTA

The concept of residual color transform is implemented in the KTA software, as of version 1.9 [40]. The flag `ResidueTransformFlag` is a part of the encoder configuration file of the KTA software. The residual color transform can be applied to encode a YUV sequence by using this flag as follows:

```
ResidueTransformFlag  = 1
# (0: no residue color transform 1: apply residue color transform)
```

3 Coding tools incorporated in KTA software

As of the version 1.2, the coding tools incorporated in the KTA software are listed as below [42]:

- Adaptive interpolation filter
- Motion compensated prediction with 1/8-pel motion vector resolution
- Motion vector competition
- Adaptive prediction error coding in spatial and frequency domain
- Adaptive quantization matrix selection

3.1. Adaptive interpolation filters:

The coding standards developed by the ISO and ITU apply hybrid video coding with motion-compensated prediction. It is combined with transform coding of prediction error [43]. First, the motion-compensated prediction is performed to reduce the temporal redundancy i.e. correlation between the previous and the current frames. After this, the prediction signal is transform coded to reduce the spatial redundancy. While performing motion-compensated prediction, the current image is divided into blocks. A displacement vector is estimated for each block. H.264/AVC is based on ¼ pel displacement resolution [43]. The reference image needs to be interpolated on sub-pel position for the estimation and compensation of fractional-pel displacement vectors. H.264/AVC uses a 6-tap Wiener interpolation filter [43]. The filter coefficients to interpolate the luma values at half-pel sample position are $(1,-5,20,20,-5,1)$. Fig. 1 illustrates the integer pel positions and fractional pel positions obtained by interpolation. First, half-pel positions of aa, bb, b, hh, ii, jj and cc, dd, h, ee, ff, gg are estimated using a horizontal or vertical 6-tap Wiener filter, respectively [43]. Then, the sub-pel position j is calculated using the same Weiner filter at positions aa, bb, b, hh, ii, jj. Then using a bilinear filter, the residual quarter-pel positions are calculated.

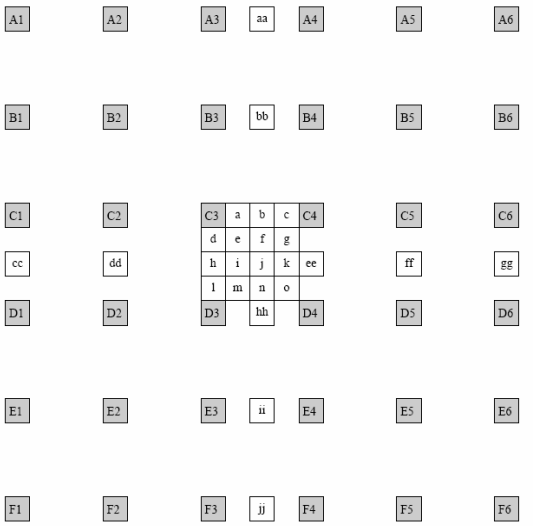

Fig. 1: Integer pel positions (shaded blocks) and fractional pel positions (white blocks)

In the context of prediction with fractional-pel motion vector resolution, the aliasing components contained in image signal limit the prediction accuracy obtained by motion compensation [43]. In order to consider aliasing, quantization and motion estimation errors, blurring effects, camera noise, etc., a two-dimensional (2D) non-separable interpolation filter, which is calculated for each P- or B- frame independently by minimizing the prediction error energy, is developed. For every fractional-pel position to be interpolated, an individual set of 2D filter coefficients is determined.

3.2. Motion compensated prediction with 1/8-pel motion vector resolution:

An image signal is interpolated on sub-pel positions using interpolation filters. H.264/AVC supports up to quarter-pel resolution for motion compensation. KTA software, as of version 1.2, supports $\frac{1}{8}$ pel resolution for motion vector. Fig. 2 illustrates interpolation process to obtain $\frac{1}{8}$ pel resolution.

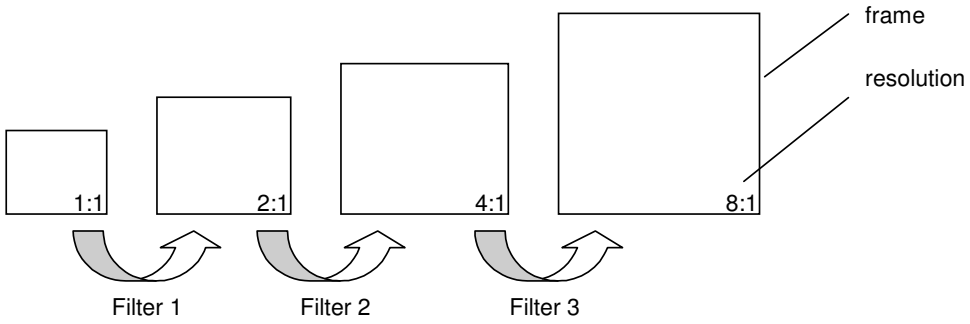

Fig. 2: Interpolation process to obtain $\frac{1}{8}$ pel resolution [44]

The interpolation process is equivalent to upsampling process. Filter 1 upsamples the given image to give 2:1 resolution. Filter 2 produces 4:1 resolution and filter 3 produces 8:1 resolution image. Several filter combinations can be used to perform this interpolation.

3.3. Motion vector competition:

H.264/AVC standard successfully achieves higher compression than its preceding standards. However, lots of bits in a H.264 stream are dedicated to motion information. Motion vector competition helps to reduce motion information [46]. A competition between spatial and temporal predictors for the motion vectors of both Inter and Skip mode is implemented. Several basic predictors are available at the encoder and at the decoder. The best predictor is selected based on the RD criterion. The index of the predictors is sent in the bitstream, if the values of the predictors are not all identical. The scheme is applied to P and B slices, which can support different set of predictors.

1.3.4. Adaptive prediction error coding in frequency and spatial domains:

H.264/AVC uses hybrid video coding where motion compensated prediction is followed by transform coding. Transform coding is efficient if the prediction error is

correlated. If the prediction error is marginally correlated then transform becomes inefficient [47]. Therefore, the prediction error is adaptively coded in the spatial or frequency domain.

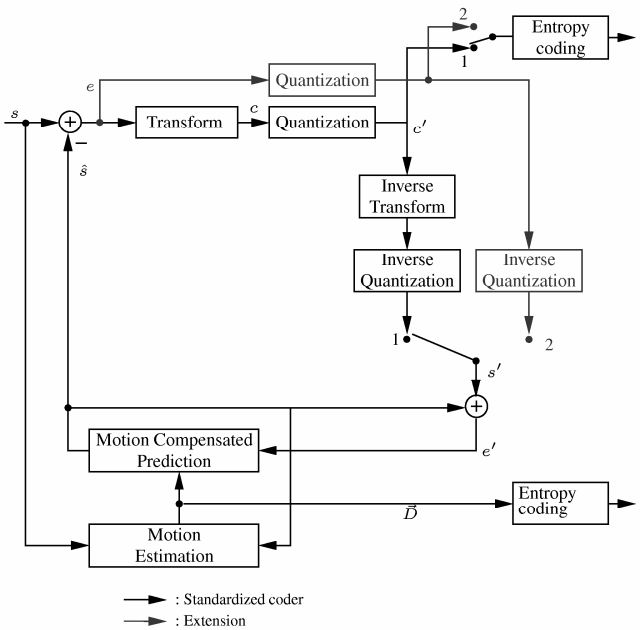

Fig. 3: Adaptive prediction error coding in spatial and frequency domains [47]

The algorithm with lower rate-distortion costs is chosen.

3.5. Adaptive quantization matrix selection (AQMS):

This tool optimizes the quantization matrix at a macroblock level. Rate distortion optimization (RDO) criterion is used to select the best quantization matrix index for a given macroblock. This index is transmitted to the decoder to perform inverse quantization using identical matrix as the encoder. The encoder and decoder block diagrams are shown in Fig. 4.

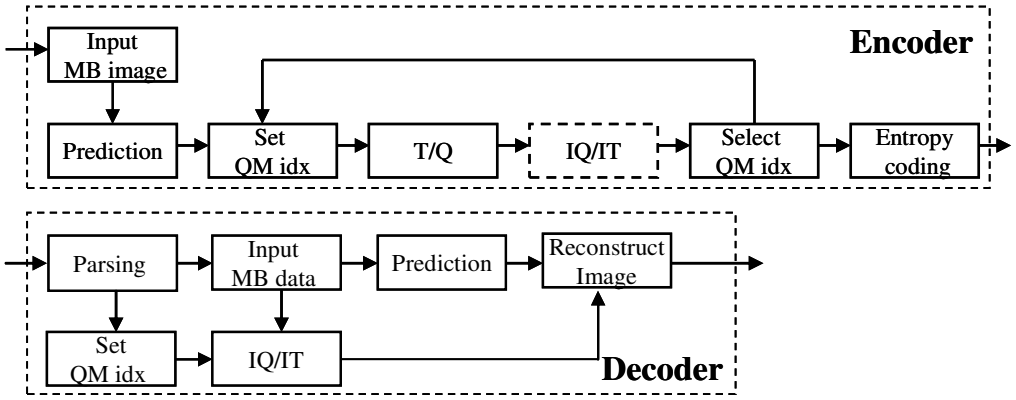

Fig. 4. Encoder and decoder block diagram for implementation of AQMS [48]

REFERENCES

[1] Soon-kak Kwon, A. Tamhankar and K.R. Rao, "**Overview of H.264 / MPEG-4 Part 10"**, J. Visual Communication and Image Representation, vol. 17, pp.183-216, April 2006.

[2] T. Wiegand and G. J. Sullivan, "**The H.264 video coding standard**", IEEE Signal Processing Magazine, vol. 24, pp. 148-153, March 2007.

[3] D. Marpe, T. Wiegand and G. J. Sullivan, "**The H.264/MPEG-4 AVC Standard and its applications**", IEEE Communications Magazine, vol. 44, pp. 134-143, Aug. 2006.

[4] D. Marpe and T. Wiegand, "**H.264/MPEG4-AVC Fidelity Range Extensions: Tools, Profiles, Performance, and Application Areas**", Proc. IEEE International Conference on Image Processing 2005, vol. 1, pp. I - 596, 11-14 Sept. 2005.

[5] A. Puri et al, "**Video Coding using the H.264/ MPEG-4 AVC compression standard**", Signal Processing: Image Communication, vol. 19, pp: 793 – 849, Oct. 2004.

[6] J. Ostermann et al, "**Video coding with H.264/AVC: Tools, Performance, and Complexity**", IEEE Circuits and Systems Magazine, vol. 4, Issue 1, pp. 7 – 28, First Quarter 2004.

[7] G. Sullivan, P. Topiwala and A. Luthra, "**The H.264/AVC Advanced Video Coding Standard: Overview and Introduction to the Fidelity Range Extensions**", SPIE conference on Applications of Digital Image Processing XXVII, vol. 5558, pp. 53-74, Aug. 2004.

[8] D. Marpe et al, "**Macroblock-adaptive residual color space transforms for 4:4:4 video coding"**, Proc. IEEE International Conference on Image Processing (ICIP 2006), pp. 3157-3160, Atlanta, GA, USA, Oct. 8-11, 2006.

[9] W. S. Kim: **Residue color transform**, JVT-L025, 12th meeting: Redmond, WA, USA, 17-23 July, 2004.
http://ftp3.itu.int/av-arch/jvt-site/2004_07_Redmond/JVT-L025.doc

[10] W. S. Kim: **Adaptive residue transform and sampling**, JVT-K018, 11th meeting: Munich, Germany, 15-19 March, 2004.
http://ftp3.itu.ch/av-arch/jvt-site/2004_03_Munich/JVT-K018.doc

[11] Y. L. Lee: **Lossless intra coding for improved 4:4:4 coding in H.264/MPEG-4 AVC**, JVT-P016, 16th meeting: Poznan, Poland, 24-29 July, 2005, http://ftp3.itu.ch/av-arch/jvt-site/2005_07_Poznan/JVT-P016.doc

[12] Y. L. Lee: **Lossless coding for professional extensions**, JVT-L017, 12th meeting: Redmond, WA, USA, 17-23 July, 2004.
http://ftp3.itu.int/av-arch/jvt-site/2004_07_Redmond/JVT-L017.doc

[13] W. S. Kim: **Advanced residual color transform**, JVT-Q059, 17th meeting: Nice, France, 14-21 October, 2005.
http://ftp3.itu.ch/av-arch/jvt-site/2005_10_Nice/JVT-Q059-L.doc

[14] JM reference software manual and software - http://iphome.hhi.de/suehring/tml/

[15] Overview of H.264, http://en.wikipedia.org/wiki/H.264

[16] YUV formats, http://www.fourcc.org/

[17] Presentation on "YCgCo Residual Color Transform", http://www-ee.uta.edu/dip/

[18] DPX to YUV converter - http://www.fastvdo.com/DPX2YUV.html

[19] High definition sequences - ftp.tnt.uni-hannover.de

[20] YUV color space – www.wikipedia.org

[21] K. Sayood, "**Introduction to Data compression"**, III edition, Morgan Kauffmann publishers, 2006.

[22] I. E.G. Richardson, **"H.264 and MPEG-4 video compression: video coding for next-generation multimedia",** Wiley, 2003.

[23] K. R. Rao and P. C. Yip, "**The transform and data compression handbook**", Boca Raton, FL: CRC press, 2001.

[24] R. C. Gonzalez, R. E. Woods, S. L. Eddins, **"Digital Image Processing Using MATLAB",** Pearson Prentice Hall, 2003

[25] S. Srinivasan et al., **Windows media video 9: overview and applications**, Signal Processing: Image Communication, volume 19, issue 9, pp. 851–875, October 2004

[26] W. Gao et al., **AVS - The Chinese next-generation video coding standard**, NAB 2004, Las Vegas, 2004.

[27] H. Malvar and G. Sullivan, **YCoCg-R: A Color Space with RGB Reversibility and Low Dynamic Range**, Document: JVT-I014r3, JVT PExt Ad Hoc Group Meeting: 22-24 July 2003, Trondheim, Norway.

[28] JPEG reference software: ftp://ftp.simtel.net/pub/simtelnet/msdos/graphics/jpegsr6.zip

[29] JPEG2000 reference software, Jasper software, Version 1.900.0: http://www.ece.ubc.ca/mdadams/jasper

[30] M.J. Weinberger, G. Seroussi and G. Sapiro, **The LOCO-I lossless image compression algorithm: principles and standardization into JPEG-LS**, IEEE Trans. Image Processing, vol. 9, pp. 1309-1324, Aug.2000. http://www.hpl.hp.com/loco/

[31] T. Wiegand et. al, **Overview of the H.264/AVC Video Coding Standard**, IEEE Trans. CSVT, Vol. 13, pp. 560-576, July 2003.

[32] A. Bovik, **Handbook of Image and Video Processing**, second edition, Elsevier Academic Press, 2005.

[33] J. Billings, **SMPTE VC1,** The IEE 2-Day Seminar on IT to HD: Visions of Broadcasting in the 21st Century, pp. 101 – 110, 30 Nov.-1 Dec. 2004

[34] R. Calderbank et al, **Wavelet transforms that map integers to integers,** Applied Computational Harmonics Analysis, 5, no. 3, pp. 332-369, 1998.

[35] D. S. Taubman and M. W. Marcellin, **JPEG2000: Image Compression Fundamentals, Standards, and Practice**, Boston, MAL Kluwer Academic Publishers, 2002.

[36] Chroma subsampling website: http://en.wikipedia.org/wiki/Chroma_subsampling

[37] K. R. Rao and J. J. Hwang, **Techniques and standards for image, video and audio coding**, Prentice Hall, 1996.

[38] Mathworks technical notes: 3http://www.mathworks.com/support/tech-notes/1100/1107.html

[39] J. Jung, **Performance evaluation of the KTA 1.2 software**, VCEG-AE09, ITU - Telecommunications Standardization Sector, STUDY GROUP 16 Question 6, Video Coding Experts Group (VCEG), 31st Meeting: Marrakech, MA, 15-16 January, 2007

[40] KTA software download: http://iphome.hhi.de/suehring/tml/download/KTA/

[41] VCEG document: http://ftp3.itu.ch/av-arch/video-site/0801_Ant/

[42] J. Jung, **KTA 1.2 software manual**, VCEG-AE08, ITU - Telecommunications Standardization Sector, STUDY GROUP 16 Question 6, Video Coding Experts Group (VCEG), 31st Meeting: Marrakech, MA, 15-16 January, 2007

[43] Y. Vatis et al, **Coding of coefficients of two-dimensional non-separable adaptive Wiener interpolation filter**, Visual Communications and Image Processing, Proceedings of the SPIE, Volume 5960, pp. 623-631, 2005

[44] T. Wedi, **1/8-pel motion vector resolution for H.26L**, ITU-T Q.15/SG16, doc. Q15-K-21, Portland, Oregon USA, August 2000

[45] J. Jung, G. Laroche, **Competition-Based Scheme for Motion Vector Selection and Coding**, VCEG Contribution VCEG-AC06, Klagenfurt, July 2006

[46] J. Jung et al, **A spatio-temporal competing scheme for the rate-distortion optimized selection and coding of motion vectors**, EUSIPCO'06, Firenza, Italy, September 2006

[47] M. Narroschke, H.G. Musmann, **Adaptive prediction error coding in spatial and frequency domain with a fixed scan in the spatial domain**, ITU-T Q.6/SG16, doc. VCEG-AD07, Hangzhou, China, October 2006

[48] A. Tanizawa, T. Chujoh, **Simulation results of AQMS on KTA software**, ITU-T SG16/Q.6 Doc. VCEG-AC07, Klagenfurt, July 2006

[49] H. Malvar and G. Sullivan, **Transform, Scaling & Color Space Impact of Professional Extensions**, JVT-H031r2, Geneva, Switzerland, 23-27 May, 2003

[50] JPEG-LS reference software: http://www.hpl.hp.com/loco/

[51] JPEG References: http://www-ee.uta.edu/dip/

BIOGRAPHICAL INFORMATION

Pooja Vasant Agawane has completed Bachelors in Electronics Engineering from V.J.T.I., Mumbai University, India in June 2006. She is currently pursuing a master's degree in the electrical engineering department at the University of Texas at Arlington. Pooja is a member of the multimedia processing research group under the guidance of Dr. K. R. Rao. She was an intern and is currently employed by the Digital Home Group, Intel Corporation, Chandler, Arizona. Her research interests include audio and video processing and digital signal processing.