

COMPLEXITY REDUCTION FOR VP6 TO H.264
TRANSCODER USING
MOTION VECTOR
REUSE

by

JAY R PADIA

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

MAY 2010

Copyright © by Jay R Padia 2010

All Rights Reserved

ACKNOWLEDGEMENTS

The successful completion of this thesis is due to the invaluable guidance, support and inspiration from a number of people.

I would like to express my profound gratitude to my advisor Dr. K. R. Rao for his continuous support, encouragement, supervision and suggestions throughout this research work. His motivation and guidance played an important role in the completion of this thesis.

I would like thank On2 Technologies and Jim Bankoski, Senior Vice President of Core Technologies and Chief Technology Officer, On2 Technologies for providing the license to use the VP6 codec software for this research.

I would like to sincerely thank Dr. Manry and Dr. Lu for taking interest in my work and agreeing to be on my thesis defense committee. I would like to thank Dr. Orintara, Dr. Losh and Dr. Sajal Das for their excellent teaching in the courses which helped me in my research work. I would also like to thank Dr. Subhashis Chaudhary of IIT Bombay whose course on Digital image processing introduced me to the domain of image and video processing.

I would like to thank the members of my team at Intel Corporation for giving me an opportunity to work in the industry and guide me during my internship. I would like to thank the members of Multimedia processing laboratory for being available for any guidance sought.

I would like to thank my friends – Harnish and Sreejana for constantly pushing me to complete my research work and this thesis.

And most importantly, I would like to thank my family for always being on my side and motivating me for achieving excellence in all my endeavors including this thesis and research work.

April 16, 2010

ABSTRACT

COMPLEXITY REDUCTION FOR VP6 TO H.264 TRANSCODER USING MOTION VECTOR REUSE

Jay R Padia, MS

The University of Texas at Arlington, 2010

Supervising Professor: K. R. Rao

VP6 is a video coding standard developed by On2 Technologies, Inc. It is the preferred codec for Macromedia Flash 8 video. VP6 gained importance with Macromedia Flash emerging as a widely adopted video streaming technology over the internet. H.264 is currently one of the most widely accepted video coding standards in the industry. It enables high quality video at low bitrates. Adobe adopted H.264 for its Flash video in August 2007. Adobe looks at adoption of H.264 as a major step towards enabling high quality video on the web. Hence there is increasing importance of techniques which can convert video from VP6 to H.264 and thereby enable high quality video transmission over the internet using Flash.

VP6 and H.264 are modern standards bearing a lot of similarities and dissimilarities. While H.264 is a complex coding standard with sophisticated coding tools to reduce the bitrate as much as possible without compromising on the quality, VP6 is much simpler and less strenuous on the processor. The VP6 standard is not able to meet the quality levels of H.264 at given bitrates; but the perceived image quality is still very good. VP6 also makes use of up to

quarter-pixel accuracy motion vectors like H.264 albeit with fewer block sizes for motion estimation. Making re-use of these motion vectors available from the decoded VP6 file can reduce the underlying encoding complexity in the transcoder to a very large extent. Also the corresponding compromise on quality is very less. The proposed technique as shown in this research can bring down the complexity significantly without much compromise in the transcoded video quality.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS	iii
ABSTRACT	iv
LIST OF ILLUSTRATIONS.....	x
LIST OF TABLES	xv
Chapter	Page
1. INTRODUCTION.....	1
1.1 Significance of video compression and standardization	1
1.2 Why is video transcoding important?	2
1.3 Requirement and usefulness of VP6 to H.264 transcoder.....	3
1.4 Outline of the work	4
2. H.264 VIDEO CODING STANDARD	6
2.1 Introduction.....	6
2.2 Profiles and levels of H.264	7
2.2.1 Profiles in H.264.....	7
2.2.1.1 Baseline profile.....	8
2.2.1.2 Main profile.....	9
2.2.1.3 Extended profile	9
2.2.1.4 High Profiles defined in the FRExts amendment	9
2.2.2 Levels in H.264	12
2.3 H.264 Encoder	12
2.3.1 Intra-prediction	14
2.3.2 Inter-prediction	16

2.3.3 Transform coding	18
2.3.4 Deblocking filter.....	20
2.3.5 Entropy coding	22
2.3.6 B-slices and adaptive weighted prediction.....	23
2.4 H.264 Decoder	25
3. VP6 VIDEO CODING STANDARD	26
3.1 Introduction.....	26
3.2 Comparison with previous flash codec	26
3.3 VP6 algorithm fundamentals	33
3.4 Coding profiles in VP6.....	35
3.5 Type of frames	35
3.5.1 Golden frames.....	36
3.6 MB modes	38
3.6.1 MB Modes in I-frames (Intra-mode).....	38
3.6.2 MB Modes in P-frames (Inter-modes and Intra-mode)	38
3.6.2.1 Nearest and Near blocks	38
3.7 Motion vectors	40
3.7.1 Encoding	41
3.7.2 Prediction loop filtering.....	41
3.7.3 Filtering for fractional pixel motion compensation.....	41
3.7.3.1 Bilinear filtering.....	42
3.7.3.2 Bicubic filtering	43
3.7.4 Support for unrestricted motion vectors	47
3.8 Prediction loop filtering	48
3.9 DCT, scan orders and coefficient token set.....	50
3.9.1 Scan order.....	50

3.9.1.1 Default scan order	51
3.9.1.2 Custom scan order	51
3.9.2 DCT encoding and coefficient token set	52
3.9.2.1 DC prediction	52
3.9.2.2 Coefficient token set.....	54
3.9.2.3 DC decoding	55
3.9.2.4 AC decoding.....	56
3.9.2.5 Decoding zero runs	57
3.10 Quantization	58
3.11 Entropy coding	59
3.11.1 Use of context information	60
3.11.2 Huffman coder.....	61
3.11.3 BoolCoder	61
4. COMPARISON BETWEEN VP6 AND H.264 STANDARDS	62
4.1 Introduction.....	62
4.2 Comparison of features and coding tools	62
4.3 Performance comparison	65
4.3.1 Encoding Time	67
4.3.2 Perceived Image quality.....	67
4.3.3 Playback smoothness	71
4.3.4 Energy efficiency in playback.....	72
4.4 Quality comparison	73
4.5 Summary.....	82
5. TRANSCODING	83
5.1 Introduction.....	83

5.2 Video transcoding architectures.....	90
5.2.1 Cascaded decoder and encoder model.....	90
5.2.2 Open loop transcoding architecture.....	92
5.2.2.1 Picture drift errors.....	93
5.2.2.2 Truncation of high-frequency coefficients.....	94
5.2.2.3 Re-quantization to reduce bitrate.....	94
5.2.3 Spatial domain transcoding architecture.....	95
5.2.4 Frequency domain transcoding architecture.....	96
5.2.5 Spatial resolution reduction.....	97
5.2.5.1 Filtering and subsampling.....	98
5.2.5.2 Pixel averaging.....	98
5.2.5.3 Discarding high frequency DCT coefficients.....	99
5.2.5.4 Motion vector composition and refinement.....	99
5.2.5.5 MB coding mode decision.....	101
5.2.6 Temporal resolution reduction.....	102
5.2.6.1 Bilinear Interpolation.....	103
5.2.6.2 Forward Dominant Vector Selection (FDVS).....	103
5.2.6.3 Telescopic Vector Composition (TVC).....	103
5.2.6.4 Activity-Dominant Vector Selection (ADVS).....	103
6. IMPLEMENTATION, RESULTS AND CONCLUSIONS.....	105
6.1 Cascaded decoder and encoder.....	105
6.2 Proposed technique – reuse of motion information from VP6.....	119
6.3 Conclusions and future work.....	129
REFERENCES.....	131
BIOGRAPHICAL INFORMATION.....	136

LIST OF ILLUSTRATIONS

Figure	Page
2.1 Different profiles in H.264 with distribution of various coding tools among the profiles	8
2.2 Tools introduced in FRExts and their classification under the new high profiles	10
2.3 H.264 Encoder block diagram	14
2.4 4x4 Luma prediction (intra-prediction) modes in H.264	15
2.5 16x16 Luma prediction modes (intra-prediction) in H.264	15
2.6 Macroblock partitioning in H.264 for interprediction (a) (L-R) 16x16, 8x16, 16x8, 8x8 blocks (b) (L-R) 8x8, 4x8, 8x4, 4x4 blocks	17
2.7 Integer and sub-pixel motion vectors; H.264 supports up to quarter pixel resolution motion vectors	17
2.8 Motion compensated prediction with multiple reference frames	18
2.9 (a) DC coefficients of 16 4x4 luma blocks (b) Matrix H1 (f) is applied to 4x4 block of DC coefficients (a) (c) Matrix H2 (f) (4x4 Hadamard transform) applied to result of figure 2.9 (b)	19
2.10 (a) DC coefficients from two 8x8 chroma blocks (b) Matrix H3 (f) (2x2 Hadamard transform) applied to chroma DC coefficients (d) (c) Matrices H1, H2 and H3 of the three transforms used in H.264	20
2.11 Boundaries in a macroblock to be filtered (luma boundaries shown with solid lines and chroma boundaries shown with dotted lines)	21
2.12 Schematic block diagram of CABAC	22
2.13 (a) Bidirectional prediction (b) Bidirectional mode with linear combination of past and future macroblock prediction signal (c) Multiple reference frame mode with linear combination of two past macroblock prediction signals	24
2.14 H.264 Decoder block diagram	25
3.1 Saturation of colors (a) VP6 – true to original (b) Flash MX - oversaturation of colors	27

3.2 (a) VP6 – Better quality picture (b) Flash MX - Blockiness of the subject and background.....	28
3.3 (a) VP6 – Better picture quality (b) Flash MX - Loss of fine details in the background.....	28
3.4 Blocking artifacts (a) VP6 – Better quality picture (b) Flash MX – Blocky artifacts can be visible in the subject and background.....	29
3.5 (a) VP6 (b) Flash MX - Artificial details can be observed	29
3.6 Low contrast backgrounds (a) VP6 – clear and sharp picture (b) Flash MX – quality deteriorates.....	30
3.7 Low contrast background (a) VP6 – Image details maintained (b) Flash MX – The reef in the background even loses its identity due to blockiness.....	30
3.8 Two pass encoding (a) VP6 – Better quality image (b) Flash MX	31
3.9 Low contrast background image (2 pass encoding) (a) VP6 (b) Flash MX	31
3.10 Low contrast background (2 pass encoding) (a) VP6 – Better quality (b) Flash MX – Blocky image.....	32
3.11 (a) VP6 (b) Flash MX - Fish in background almost appear like artifacts of low contrast ocean background.....	32
3.12 Absolute loss of visual information (2 pass encoding) (a) VP6 – Quality maintained (b) Flash MX image	33
3.13 VP6 – (a) Encoder block diagram (b) Decoder block diagram	34
3.14 Previous frame prediction	36
3.15 Golden frame prediction (a) Golden frame buffer has the default i.e. last decoded I-frame (b) Golden frame buffer is updated.....	37
3.16 Order of the adjacent blocks to find Near and Nearest neighbors	39
3.17 Support for motion vector beyond the image boundaries	48
3.18 Prediction loop filtering of 8x8 block boundaries.....	49
3.19 Default zig-zag scan order	51
3.20 DC prediction using neighbors	53
3.21 Binary coding tree for AC and DC contexts	55
4.1 An early frame from H.264	68

4.2 VP6 encoded with CBR.....	68
4.3 VP6 encoded with VBR.....	69
4.4 H.264 shows better quality even as VP6 starts breaking significantly.....	70
4.5 VP6 CBR encoded frame show significant blockiness.....	70
4.6 VP6 VBR frame is better that CBR frame but fails to match H.264.....	71
4.7 Foreman clip (1st frame) (a) VP6 (b) original (c) H.264 (PSNR in dB).....	75
4.8 Akiyo clip (1st frame) (a) VP6 (b) original (c) H.264 (PSNR in dB).....	76
4.9 Akiyo clip (30th frame) (a) VP6 (b) original (c) H.264 (PSNR in dB).....	76
4.10 Average Y component MSE vs. bitrate for the Akiyo sequence.....	77
4.11 Average Y component PSNR vs. bitrate for Akiyo sequence.....	77
4.12 Average Y component SSIM vs. bitrate for Akiyo sequence.....	78
4.13 2nd frame from the Stefan (CIF) clip (PSNR in dB).....	79
4.14 15th frame from the Stefan (CIF) clip (PSNR in dB).....	80
4.15 Average Y component MSE vs. bitrate for the Stefan (CIF) clip.....	81
4.16 Average Y component PSNR vs. bitrate for Stefan (CIF) clip.....	81
4.17 Average Y component SSIM vs. bitrate for the Stefan clip.....	82
5.1 Communication between various multimedia devices.....	83
5.2 Video transcoding operations.....	84
5.3 Video coding standards (a) Timeline [43] (b) Increase in complexity.....	86
5.4 Video transcoding operations and classification.....	87
5.5 Various ways of spatial transcoding (Bits = target bitrate in kbps).....	88
5.6 Transcoding with normal downsampling and user-interest-based downsampling.....	89
5.7 Cascaded decoder and encoder model (a) block level diagram (b) detailed diagram.....	91
5.8 Open loop transcoder architecture.....	92

5.9 Bitrate reduction by truncation of high frequency coefficients.....	94
5.10 Transcoding with re-quantization scheme	95
5.11 Spatial domain transcoding architecture (SDTA) with MV reuse and STR.....	96
5.12 Simplified SDTA without STR	96
5.13 Frequency domain transcoding architecture (FDTA)	97
5.14 Decimation of a 16 x 16 MB	98
5.15 Pixel averaging.....	98
5.16 DCT domain decimation for SRR.....	99
5.17 Four motion vectors being downsampled to one	99
5.18 Motion vector refinement using search window (a) best case – small search (b) worst case – long search.....	101
5.19 Four macroblock types downsampled to one	102
5.20 FDVS motion vector composition scheme for TRR	103
6.1 Cascaded decoder and encoder	105
6.2 Foreman sequence (bitrate ~ 10 kbps) – frame 1 (a) original sequence (b) cascaded decoder and encoder (c) VP6 (d) H.264 baseline	107
6.3 Foreman sequence - Y component MSE - cascaded implementation.....	108
6.4 Foreman sequence – Y component PSNR (dB) – cascaded implementation.....	108
6.5 Foreman sequence – Y component SSIM – cascaded implementation	109
6.6 Akiyo sequence (bitrate ~ 9.3 kbps) – frame 2 (a) original sequence (b) cascaded decoder and encoder (c) VP6 (d) H.264 baseline	111
6.7 Akiyo sequence (bitrate ~ 9.3 kbps) – frame 2 (a) original sequence (b) cascaded decoder and encoder (c) VP6 (d) H.264 baseline	112
6.8 Akiyo sequence – Y component MSE – cascaded implementation.....	113
6.9 Akiyo sequence – Y component PSNR – cascaded implementation	113
6.10 Akiyo sequence – Y component SSIM – cascaded implementation.....	114
6.11 Stefan sequence – frame 1 – I-frame (a) original sequence (b) cascaded decoder and encoder (c) VP6 (d) H.264 baseline	116

6.12 Stefan sequence – frame 15 – an intermediate frame (a) original sequence (b) cascaded decoder and encoder (c) VP6 (d) H.264 baseline	117
6.13 Stefan sequence – Y component MSE – cascaded implementation	118
6.14 Stefan sequence – Y component PSNR (dB) – cascaded implementation	118
6.15 Stefan sequence – Y component SSIM – cascaded implementation	119
6.16 Proposed technique of reusing MB modes and MV from VP6	121
6.17 Foreman clip – PSNR (dB) vs. bitrate (kbps) – Cascaded architecture and proposed technique	122
6.18 (a) Foreman sequence – Y component SSIM for predicted frame 1 (b) Foreman sequence – Y component SSIM for predicted frame 2	123
6.19 Foreman clip – Motion estimation time (MET) vs bitrate (kbps) – Cascaded architecture and proposed technique	124
6.20 Akiyo clip – PSNR (dB) vs. bitrate (kbps) – Cascaded architecture and proposed technique	125
6.21 Akiyo clip – Motion estimation time (MET) vs. bitrate (kbps) – Cascaded architecture and proposed technique	126
6.22 Stefan clip – PSNR (dB) vs bitrate (kbps) – Cascaded architecture and proposed technique	127
6.23 (a) Stefan sequence – Y component SSIM vs. Bitrate (kbps) – 1st predicted frame (b) Stefan sequence – Y component SSIM vs. Bitrate (kbps) – 2nd predicted frame	128
6.24 Stefan clip – Motion estimation time (MET) vs. bitrate (kbps) – Cascaded architecture and proposed technique	129

LIST OF TABLES

Table	Page
2.1 Comparison of the high profiles and corresponding coding tools introduced in the FRExts	11
2.2 Levels defined in H.264.....	12
3.1 MB coding modes in VP6.....	39
3.2. Bilinear (1-D) filter taps for ¼ sample precision Luma filtering	42
3.3. Bilinear (1-D) filter taps for 1/8 sample precision chroma filtering	42
3.4. Bicubic (4-tap) filter tabs for 1/8 pixel interpolation	43
3.5 Prediction loop filter limit values	49
3.6 Coefficient band update probabilities	52
3.7 DC prediction based on presence of neighboring blocks.....	53
3.8 DCT token set and extra bits.....	54
3.9 DC Node contexts	56
3.10 AC coefficient bands	56
3.11 AC preceding decoded coefficient context.....	57
3.12 AC coefficient bands for Huffman	57
3.13 Zero runs coefficient bands	57
3.14 Zero runs node index	58
3.15 DC and AC quantization values	59
4.1 Comparison of different flash video standards.....	63
4.2 CPU usage comparison (in percentage) for different codecs (MacPro 8 cores)	72
4.3 Quality metrics for Foreman clip for VP6 and H.264 codecs (PSNR in dB)	74
4.4 Quality metrics for Akiyo (QCIF) clip for VP6 and H.264 (PSNR in dB)	75

4.5 Quality metrics for Stefan sequence (PSNR in dB)	78
5.1 Different multimedia applications and corresponding video standards.....	85
6.1 Foreman sequence – quality metrics for cascaded implementation (PSNR in dB)	106
6.2 Akiyo sequence – quality metrics for cascaded implementation (PSNR in dB).....	109
6.3 Stefan sequence – quality metrics for cascaded implementation (PSNR in dB)	114
6.4 Motion estimation and compensation (VP6 vs. H.264)	120
6.5 PSNR (dB) and MET for Foreman clip using cascaded architecture and the proposed technique	121
6.6 PSNR (dB) and MET for Akiyo clip using cascaded architecture and the proposed technique	124
6.7 PSNR (dB) and MET for Foreman clip using cascaded architecture and the proposed technique	126

CHAPTER 1

INTRODUCTION

1.1 Significance of video compression and standardization

We live in the age of Information and Communications technology (ICT). Storage and transmission of data at present is more important than it ever was. The emerging importance of ICT has been complemented by the digital revolution [1] [2] [3]. The advent of digital storage and transmission in the last few decades is a major event. Analog information needs light and sound waves for transmission and media such as films for storage; whereas digital information of any kind can be represented in 1s and 0s (bits). All different kinds of information such as audio, video, images, radar signals, instructions to satellites and space stations, etc. can be stored and transmitted in the same form. With this information (or data) generated in large amount and ability to process it faster and with fewer errors, data compression assumes a lot of importance. Data compression enables representation of entities such as text files, videos, etc. with reduced number of bits.

Data compression involves taking advantage of the redundancy in data to represent the information in compact form [4]. There are lossless as well as lossy compression techniques. If the data has been losslessly compressed, the original data can be recovered exactly with no loss. It is applied in areas where data loss can be detrimental. Text compression is an important area of lossless compression [4]. Lossy compression involves some loss of information; so the data cannot be recovered exactly. In exchange for such tolerable distortion much higher compression can be achieved. Speech compression can be an application [4] where loss of information such as high frequency sounds above human hearing capacity can be lost without loss of fidelity. Video compression also involves lossy compression [4].

Video has both spatial and temporal redundancy. Getting rid of this redundancy and other lossy techniques facilitate very high compression. The volume of video data is usually very high. For example, in order to digitally represent 1 second of video without compression (using CCIR 601 format), more than 20 MBytes or 160 Mbits is required [4]. This amount of data indicates the importance of compression for video signals.

Multimedia applications are targeted for a wide range of applications such as video conferencing, video on demand, mobile video broadcast and even medical applications. Given such wide range of applications standardization of video compression techniques is essential. Standardization ensures interoperability of implementation from different vendors thereby enabling end-users to access video from different services both software and hardware [1]. There are numerous video compression standards both open-source and proprietary depending on the application and end-usage. Experts from academia and industry have formed the Moving Pictures Experts Group (MPEG) and Video Coding Experts Group (VCEG) with the aim to bring standardization in coding moving pictures or video. The MPEG and VCEG joined together to form the Joint Video Team (JVT) in 2001 which developed the ITU-T Rec. H.264 | ISO/IEC 14496-10, commonly referred to as H.264/MPEG-4-AVC, H.264/AVC, or MPEG-4 Part 10 AVC [5]. VP6 is a proprietary standard from On2 Technologies, Inc. In 2005, it was adopted by Adobe for its Flash Video and included in its Adobe Flash Player [25].

1.2 Why is video transcoding important?

Different video compression standards assume significance due to the difference in the access to network connectivity, bandwidth, computational capacity, display rate, etc. available to the end-user. To be able to deliver and reproduce video and other multimedia data flexibly according to the end-user's requirements and capability, content should be dynamically adapted to the user's environment. This can include altering characteristics such as bit-rate, frame-rate, spatial resolution, coding syntax and even content. The process of transcoding plays an important role fulfilling this requirement. A video transcoder is defined as an operation of

converting video from one video format to another [30]. Transcoding also has other applications such as in statistical multiplexing of video to maintain bandwidth, to include new attributes such as company logos, watermarks, etc., adding error resilience capabilities and others [30].

1.3 Requirement and usefulness of VP6 to H.264 transcoder

TrueMotion VP6 from On2 Technologies was adopted by Adobe for its Flash suite of products [26]. This has led to emerging importance of VP6 standard and its wide outreach.

Flash video is a very important aspect of accessibility of video playback on the internet. Flash video is rapidly changing the landscape of video on the Web. It is emerging as the preferred solution for providing video services online over Windows Media Player, Apple Quicktime and Real Networks Real Player [7]. The advantages of Flash Player over its rivals are its small size and its completeness as a website development package. Its ability to support multiple platforms has made it popular [7].

In August 2007, Adobe also adopted the H.264 video coding standard in the Adobe® Flash® Player 9 software [8]. H.264 has a set of innovations which can together provide a vast improvement in performance over previous generations of video codecs. With H.264 extended to the Flash ecosystem, customers can leverage their existing video and audio to deliver content to the Web and other devices – up to HD quality. Adobe targeted development at lower costs and wider penetration by adoption of H.264 which is already a widely accepted media standard [8].

According to John Loiacono, senior Vice President of Creative Solutions at Adobe, “Already a broadly adopted industry standard, the inclusion of the H.264 codec in Adobe Flash Player, Adobe AIR, the Creative Suite® product line, and the upcoming Adobe Media Player will accelerate customer workflows, enabling the creation and repurpose of high-quality Web video content without extra development costs [8].”

The adoption of H.264 which has superior video quality over most other existing codecs, the Flash Player now will be supporting arguably the most popular video standard out

there. This promises the enabling of availability of HD video on the web to whoever who wants it. Flash Player content reaches over 98 percent of Internet-enabled desktops. More than 80 percent of online videos worldwide are viewed using Adobe Flash technology, making it the number one format for video on the Web [8]. Adoption of a previous update to Flash Player 9 set all-time records by achieving nearly 90 percent reach on Internet-enabled desktops in less than nine months.

Tools like Adobe Premier Pro and Adobe After Effects support H.264 encoding at present. As Flash Player supports playback of any H.264 encoded video developers can leverage both the existing video assets encoded as well as the entire spectrum of tools and infrastructure that support H.264 [8]. Thus Adobe has this huge ecosystem now built around the H.264 codec.

According to Adobe, the adoption of H.264 for Flash is a great thing for web video. Combination of a great format like H.264 and a runtime like Flash is the best thing to happen to web for it to embrace HD-quality video [8].

On2 Truemotion VP6 was the main codec for Flash Video. The adoption of H.264 by flash and the rapid rise in its acceptance as the ideal format for high quality web video paves the way for the proposed research. Here a transcoding technique to transcode existing VP6 content to H.264 is proposed.

1.4 Outline of the work

The research work presented here proposes a reduced complexity transcoding technique to transcode a VP6 video sequence to H.264 video sequence. Before transcoding between two codecs, it is important to understand the coding tools and syntax of both the codecs. The second chapter contains an overview of the H.264 codec. H.264 is an advanced codec with many sophisticated coding tools introduced for the first time. The third chapter describes the VP6 standard. VP6 is proprietary standard. The chapter gives an overview of the coding technique used in VP6 to achieve good quality at lower bitrates with minimal complexity.

It is important to compare two codecs before proposing a transcoding technique. Chapter four highlights the similarities and dissimilarities between the two codecs and provides a performance and output quality comparison. Chapter five gives an overview of the basic transcoding architectures, their advantages and disadvantages. The steps and technique implemented to reduce the transcoding complexity as compared to basic cascaded architecture is described in chapter six. This chapter also describes the results and conclusions along with suggestion on possible future work.

CHAPTER 2

H.264 VIDEO CODING STANDARD

2.1 Introduction

H.264/MPEG4-Part 10 advanced video coding (AVC) introduced in 2003 became one of the latest and most efficient video coding standards [9]. 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 [9].

H.264 can support various interactive (video telephony) and non-interactive applications (broadcast, streaming, storage, video on demand) as it facilitates a network friendly video representation [11]. It leverages on the previous coding standards such as MPEG-1, MPEG-2, MPEG-4 part 2, H.261, H.262 and H.263 [10] [12] and adds many other coding tools and techniques which give it superior quality and compression efficiency.

Like any other previous motion-based codecs, it uses the following basic principles of video compression [5]:

- 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.

The improved coding efficiency of H.264 can be attributed to the additional coding tools and the new features. Listed below are some of the new and improved techniques used in H.264 for the first time [11]:

- 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 [11]:

- 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.

2.2.1. Profiles in H.264

Profiles are defined as a subset of coding tools. They help to maximize the interoperability while limiting the complexity [5] [13]. 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 [14]:

1. Baseline profile
2. Extended profile
3. Main profile
4. High Profiles defined in the FRExts amendment

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

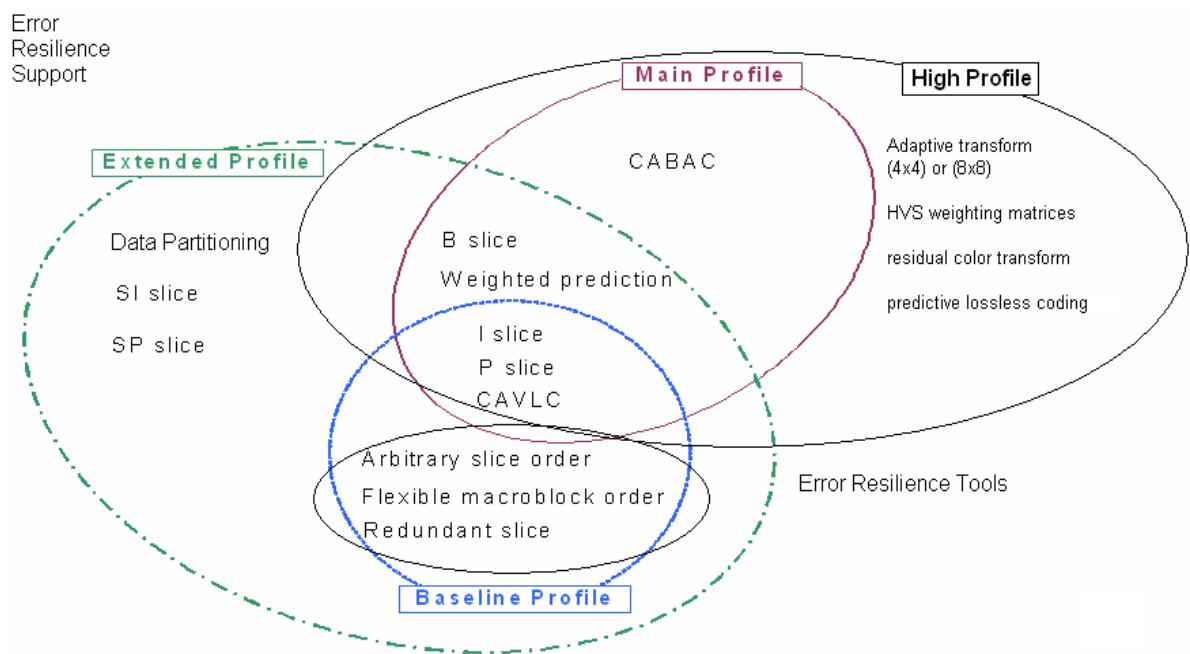


Figure 2.1 Different profiles in H.264 with distribution of various coding tools among the profiles [10]

2.2.1.1. 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, arbitrary 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.

2.2.1.2. 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.

2.2.1.3. 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.

2.2.1.4. High Profiles defined in the FRExts amendment

In September 2004 the first amendment of H.264/MPEG-4 AVC video coding standard was released [14]. A new set of coding tools were introduced as a part of this amendment. These are termed as “Fidelity Range Extensions” (FRExts). The aim of releasing FRExts is to be able to achieve significant improvement in coding efficiency for higher fidelity material. The application areas for the FRExts tools are professional film production, video production and high-definition TV/DVD.

The FRExts amendment defines four new profiles (refer Figure 2.2) [15] [17]:

- High (HP)
- High 10 (Hi10P)
- High 4:2:2 (Hi422P)
- High 4:4:4 (Hi444P)

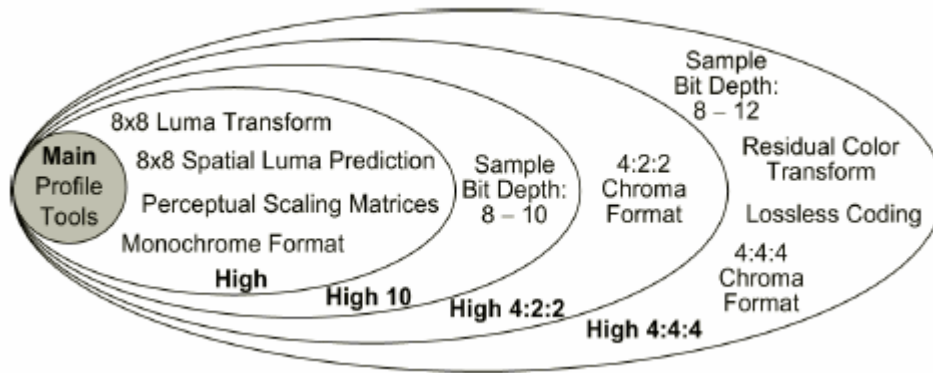


Figure 2.2 Tools introduced in FRExts and their classification under the new high profiles [14]

All four of these profiles build further upon the design of the prior Main profile. The table 2.1 provides a comparison of the high profiles introduced in FRExts with a list of different coding tools and which of them are applied to which profile. All of the high profiles include the following three enhancements of coding efficiency performance [15]:

- Adaptive macroblock-level switching between 8x8 and 4x4 transform block sizes

The main aim behind introducing 8x8 transform in FRExts was because high fidelity video demands preservation of fine details and textures. To achieve this, larger basis functions are required. However smaller transform like 4x4 reduces ringing artifacts and reduces computational complexity. The encoder adaptively chooses between 4x4 and 8x8 transform.

The transform selection process is limited by the following conditions

- If an inter-coded MB has sub-partition smaller than 8x8 (i.e. 4x8, 8x4, 4x4), then 4x4 transform has to be used.
- If an intra-coded MB is predicted using 8x8 luma spatial prediction, only 8x8 transform is used.
- 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 [10].

- Encoder-specified separate control of the quantization parameter for each chroma component

Table 2.1 Comparison of the high profiles and corresponding coding tools introduced in the FRExts [14]

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 Quantization parameter (QP) control	x	x	x	x
Monochrome video format	x	x	x	x
9 and 10 bit sample depth		x	x	x
4:2:2 Chroma format			x	x
11 and 12 sample bit depth				x
4:4:4 Chroma format				x
Residual color transform				x
Predictive lossless coding				x

2.2.2. Levels in H.264

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 interoperability for applications with similar functional requirements [16]. The levels defined in H.264 are listed in Table 2.1. The level '1b' was added in the FRExts amendment.

Table 2.2 Levels defined in H.264 [7]

Level number	Typical picture size	Typical frame rate	Maximum compression bit rate (for VLC) 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	2k x 1k	72	135 Mbps	5
5.1	2k x 1k or 4k x 2k	120 / 30	240 Mbps	5

2.3 H.264 Encoder

Fig. 2.3 illustrates the schematic of the H.264 encoder. H.264 encoder works on macroblocks and motion-compensation like most other previous generation codecs. Video is formed by a series of picture frames. Each picture frame is an image which is split down into blocks. The block sizes can vary in H.264. 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. H.264 uses nine spatial prediction modes in intra-coding to 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. The aim to use inter-coding is to reduce the temporal redundancy by making use of motion vectors. Motion vectors give the direction of motion of a particular block from the current frame to the next frame. The prediction residuals are obtained which then undergo transformation to remove spatial correlation in the block. The transformed coefficients, thus obtained, undergo quantization. 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) [10].

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 in the sections 2.3.1 thru 2.3.6.

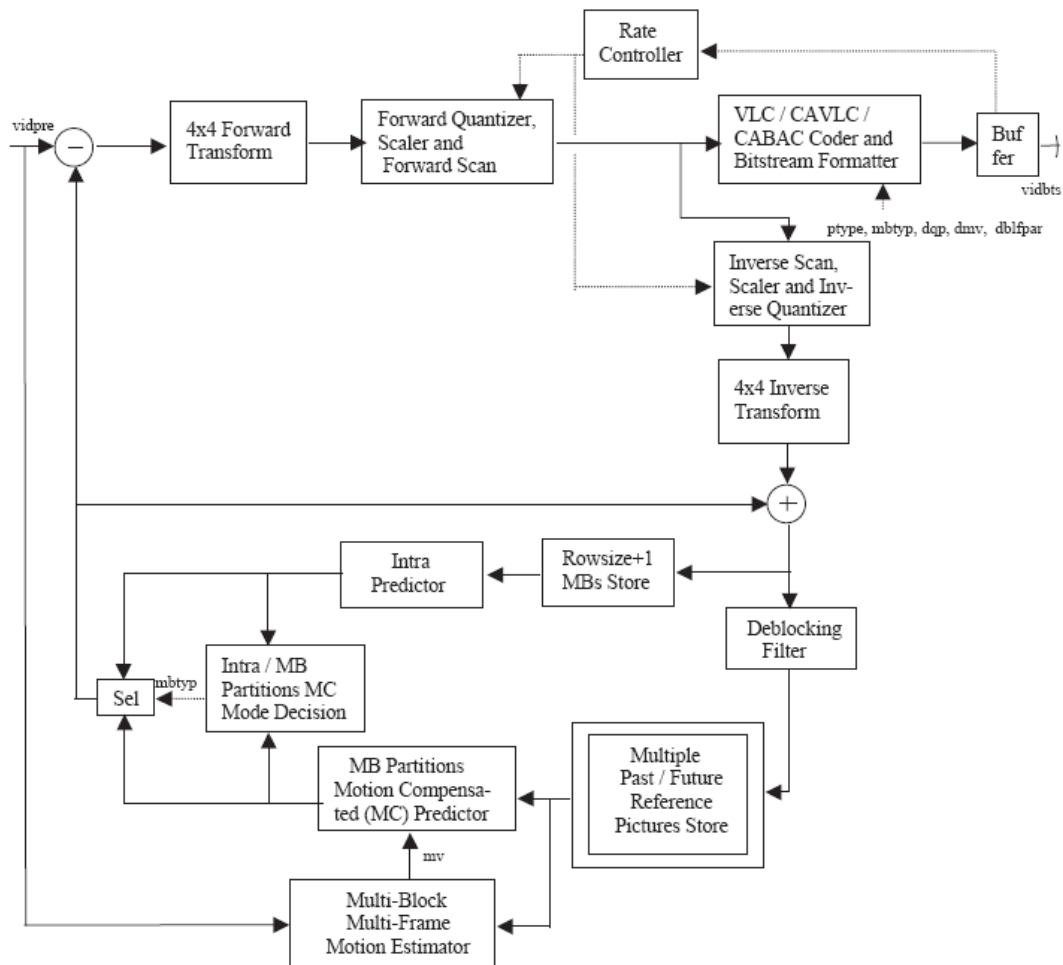


Figure 2.3 H.264 Encoder block diagram

2.3.1. Intra-prediction

Intra-prediction uses the macroblocks from the same image for prediction. Two types of prediction schemes are used for the luminance component. These two schemes can be referred as INTRA_4x4 and INTRA_16x16 [16]. 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 as shown in Fig. 2.4 [18] [19].

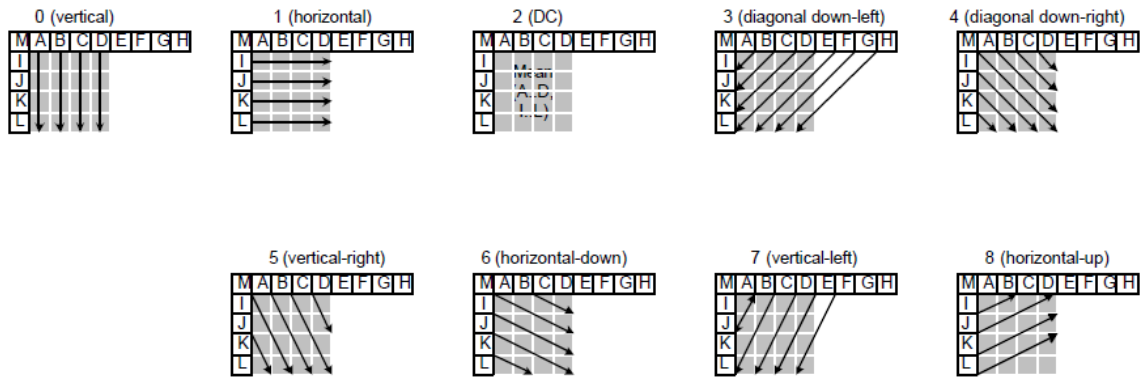


Figure 2.4 4x4 Luma prediction (intra-prediction) modes in H.264 [19]

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.

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.

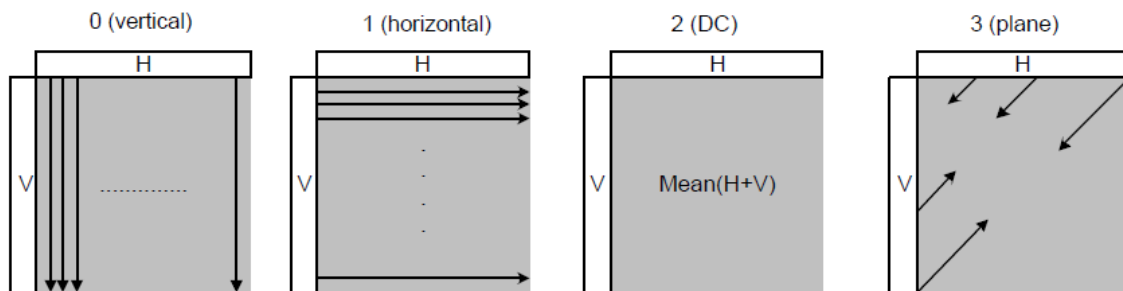


Figure 2.5 16x16 Luma prediction modes (intra-prediction) in H.264 [19]

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 [10].

2.3.2. Inter-prediction

Inter-prediction is used to capitalize on the temporal redundancy in a video sequence. The temporal correlation is reduced by inter prediction through the use of motion estimation and compensation algorithms [10]. An image is divided into macroblocks; each 16x16 macroblock is further partitioned into 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. A smaller block size ensures less residual data; however smaller block sizes also mean more motion vectors and hence more number of bits required to encode these motion vectors [18] [20].

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 t and d in the figure 2.8 are the image reference parameters.

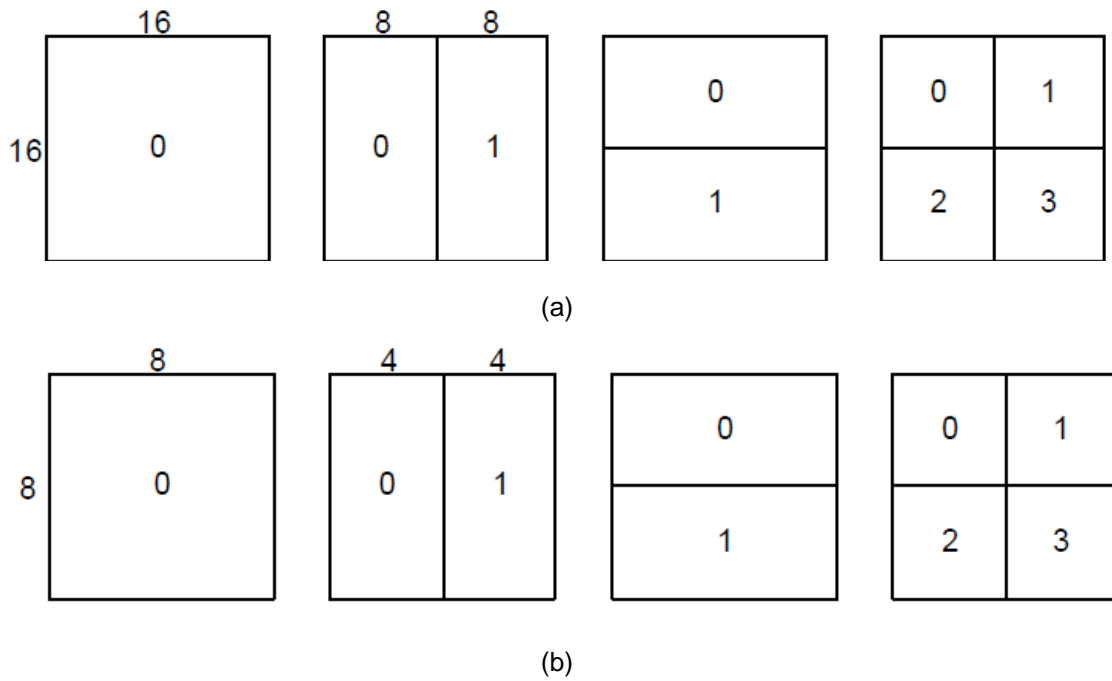


Figure 2.6 Macroblock portioning in H.264 for interprediction
 (a) (L-R) 16x16, 8x16, 16x8, 8x8 blocks
 (b) (L-R) 8x8, 4x8, 8x4, 4x4 blocks [20]

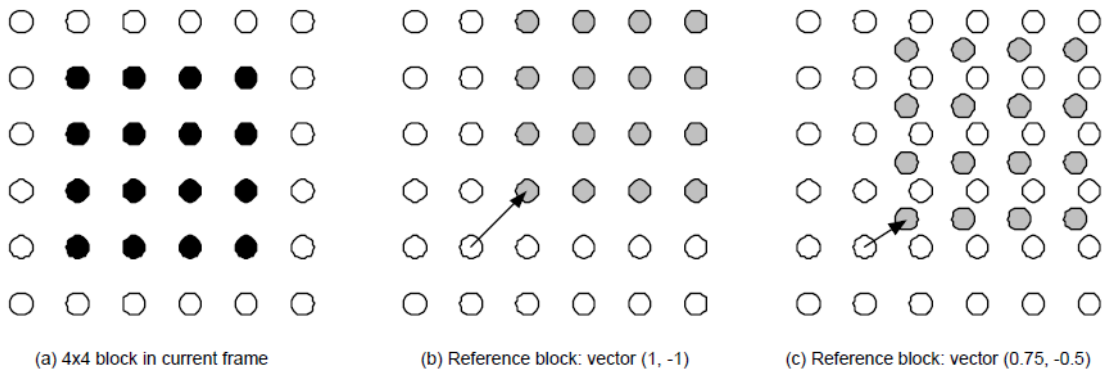


Figure 2.7 Integer and sub-pixel motion vectors; H.264 supports up to quarter pixel resolution motion vectors [20]

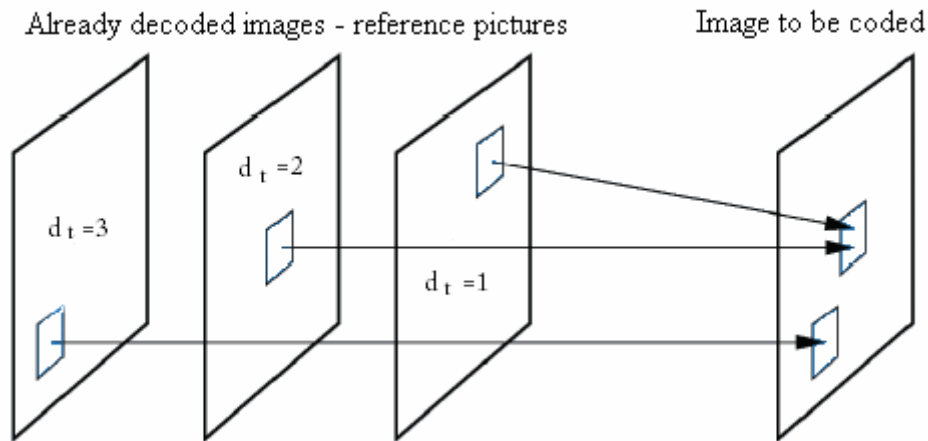


Figure 2.8 Motion compensated prediction with multiple reference frames [16]

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 [10]. The former standards of MPEG-1 and MPEG-2 employed a two dimensional discrete cosine transform (DCT) for the purpose of transform coding of the size 8×8 [16]. H.264 uses integer transforms instead of the DCT. The size of these transforms is 4×4 [16]. 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 4×4 transform has benefit for removing the need for multiplications.

H.264 employs a hierarchical transform structure, in which the DC coefficients of neighboring 4×4 transforms for luma signals are grouped into 4×4 blocks and transformed again by the Hadamard transform (figure 2.9 (a)). As shown in figure 2.9 (b) the first transform (matrix

H1 in figure 2.10 (c)) 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 H2 in Fig. 2.10 (c)) for luma DC coefficients (figure 2.9 (c)) for 16x16 intra-mode only, and 2x2 Hadamard transform as shown in figure 2.10 (a) and (b) (matrix H3 in Fig. 2.10 (c)) for chroma DC coefficients.

00	01	02	03
0	1	4	5
10	11	12	13
2	3	6	7
20	21	22	23
8	9	12	13
30	31	32	33
10	11	14	15

(a)

$$Y = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \begin{bmatrix} x_{00} & x_{01} & x_{02} & x_{03} \\ x_{10} & x_{11} & x_{12} & x_{13} \\ x_{20} & x_{21} & x_{22} & x_{23} \\ x_{30} & x_{31} & x_{32} & x_{33} \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{bmatrix}$$

(b)

$$\left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} x_{D00} & x_{D01} & x_{D02} & x_{D03} \\ x_{D10} & x_{D11} & x_{D12} & x_{D13} \\ x_{D20} & x_{D21} & x_{D22} & x_{D23} \\ x_{D30} & x_{D31} & x_{D32} & x_{D33} \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \right) // 2$$

(c)

Figure 2.9 (a) DC coefficients of 16 4x4 luma blocks [6]
 (b) Matrix H1 (2.10 c) is applied to 4x4 block of DC coefficients (a) [6]
 (c) Matrix H2 (2.10 c) (4x4 Hadamard transform) applied to result of figure 2.9 (b) [6]



(a)

$$\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} DC_{00} & DC_{01} \\ DC_{10} & DC_{11} \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

(b)

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

(c)

Figure 2.10 (a) DC coefficients from two 8x8 chroma blocks [6]
 (b) Matrix H3 (f) (2x2 Hadamard transform) applied to chroma DC coefficients (d) [6]
 (c) Matrices H1, H2 and H3 of the three transforms used in H.264 [16]

2.3.4. Deblocking filter

The deblocking filter is used to remove the blocking artifacts due to the block based encoding pattern. The transform applied after intra-prediction or inter-prediction is on blocks; the transform coefficients then undergo quantization. These block based operations are responsible for blocking artifacts which are removed by using the in-loop deblocking filter. It reduces the artifacts at the block boundaries and prevents the propagation of accumulated noise. The presence of the filter however adds to the complexity of the system [22]. Fig. 2.11 illustrates a macroblock with sixteen 4x4 subblocks along with their boundaries.

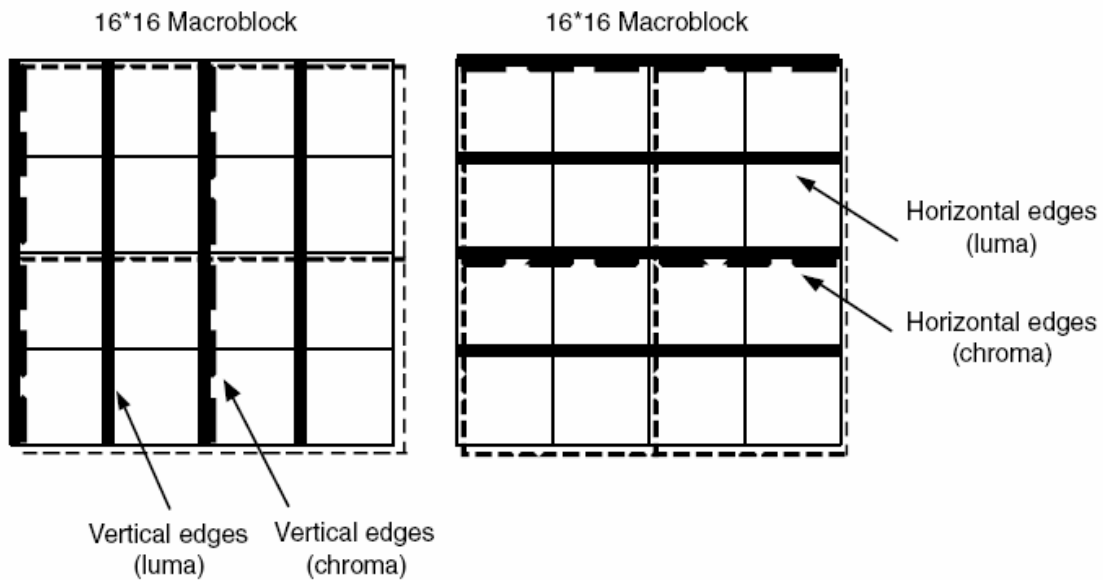


Figure 2.11 Boundaries in a macroblock to be filtered (luma boundaries shown with solid lines and chroma boundaries shown with dotted lines) [10]

As shown in the Fig. 2.11 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 a 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 [10]. 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 [10]. 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 to achieve good compression. The schematic for CABAC is shown in Fig.. 2.12.

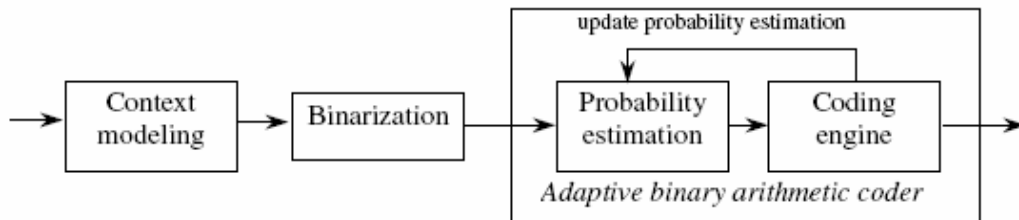


Figure 2.12 Schematic block diagram of CABAC [10]

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.

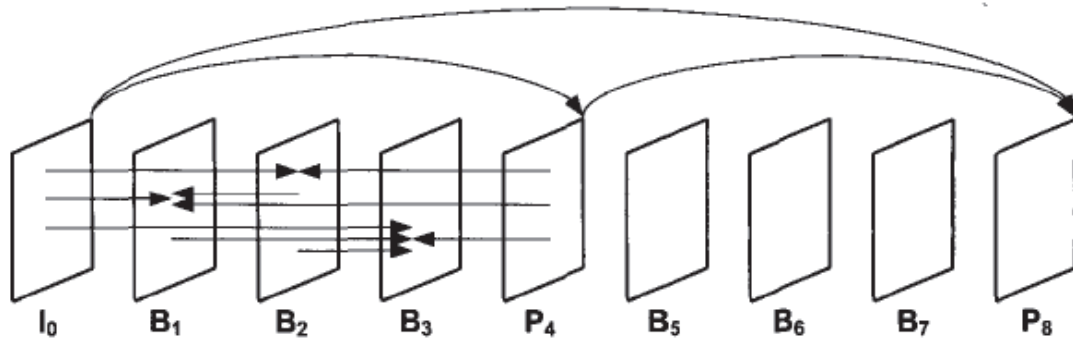
2.3.6. B-slices and adaptive weighted prediction

Bi-directional prediction where you use both past and future frames for reference can be very useful in improving the temporal prediction. Bi-directional prediction in H.264 uses multiple reference frames. Figure 2.13 (a) show bidirectional prediction from multiple reference frames. 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.

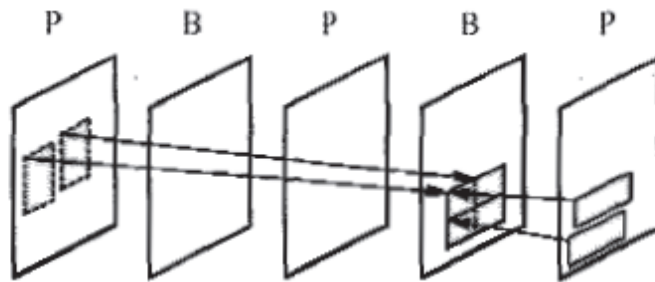
H.264 supports forward/backward prediction pair and also supports forward/forward and backward/backward prediction pair [10]. Figure 2.13 (b) and 2.13 (c) describe the scenario where bidirectional prediction and multiple reference frames respectively are applied and a macroblock is thereby predicted as a linear combination of multiple reference signals using weights as described in equation 2.1. 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 also uses weighted prediction [10]. It can be used for a macroblock of P slice or B slice. Different weights can be assigned to two different reference signals and the prediction signal is calculated as follows:

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

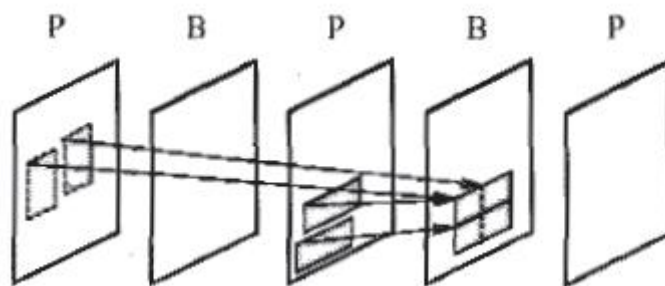
In (2.1), p is the prediction signal, r1 and r2 are the reference signals and w1 and w2 are the prediction weights.



(a)



(b)



(c)

Figure 2.13 (a) Bidirectional prediction
 (b) Bidirectional mode with linear combination of past and future macroblock prediction signal
 (c) Multiple reference frame mode with linear combination of two past macroblock prediction signals

2.4 H.264 Decoder

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 interpredicted 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.

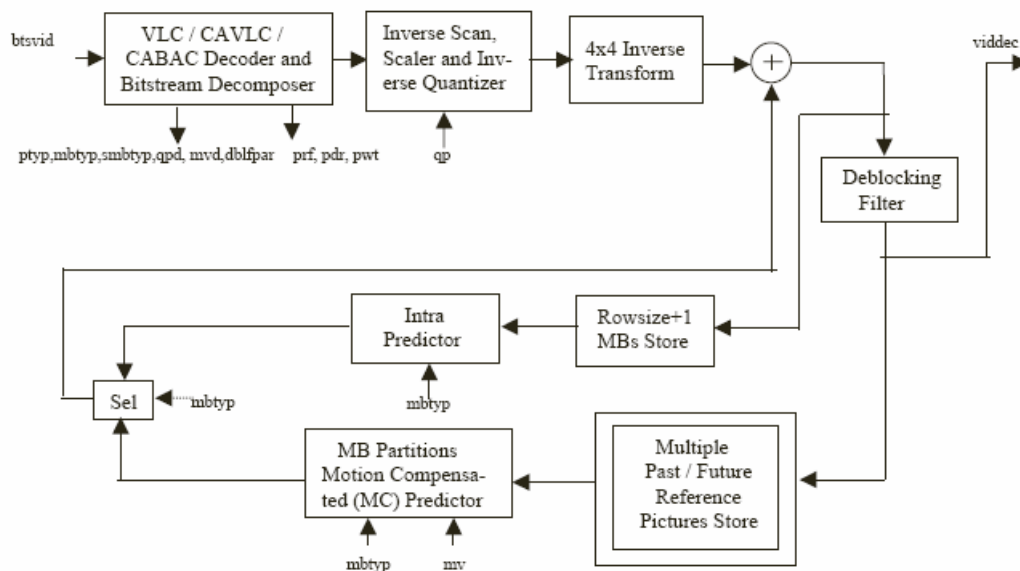


Figure 2.14 H.264 Decoder block diagram [13]

CHAPTER 3

VP6 VIDEO CODING STANDARD

3.1 Introduction

TrueMotion VP6 [25] is a new compression technology from On2 Technologies Inc. Adobe licensed it for its Flash suite of products [26]. It features as the main codec for Flash 8 and onwards. It has interesting features as it gives a very good quality at very high compression. TrueMotion VP6 is among the best video codecs on the market today. According to On2 Technologies Inc., it offers better image quality and faster decoding performance than Windows Media 9 [23], Real 9 [24], H.264 [5], and QuickTime MPEG-4 [25]. In internal testing at On2 Technologies Inc, TrueMotion VP6 could beat many H.264 implementations, Windows Media 9 and Real Networks 10 in PSNR comparisons using standard MPEG-2 test source clips [10]. The VP6 clips were more detailed and contained fewer artifacts than Windows Media 9 and maintained more texture and detail than Real or H.264 [25].

VP6.2, the latest version of TrueMotion VP6, features a significant increase in performance from the previous versions of VP6 [25].

3.2 Comparison with previous flash codec

Adobe adopted VP6 into Flash in 2005. VP6 can provide significantly better performance over the previous generation Flash codec MX which used the Sorrenson Spark codec [32] based on H.263 [33]. On2 Technologies Inc. licenses the VP6 codec. The authors in [25] provide a performance comparison between Flash MX and Flash video with VP6. The results as described in this section show an improvement in multiple aspects.

This section describes the comparative study from authors at On2 by citing data and figures from [25]. The images shown in figures 3.2 thru 3.12 are excerpts for a 12:30 minute

video of coral reef exploration. The original source was shot on DVCAM and was stored in photo-jpeg compression. Being shot from a DVCAM, the 720x486 DV source needed to have some over-scan cropped out. De-interlacing was performed on it and the source was resized to 320x240 [25].

The tool used for VP6 encoding was Flix Professional from On2 Technologies Inc. The comparison provides examples of both single pass and two pass encoding which is supported in Flix. The samples for this study were encoded a fairly low bitrate of about 150 kbps [25]. In the figures, the image on the left is from VP6 and the one on the right is from Flash MX video.



Figure 3.1 Saturation of colors (a) VP6 – true to original
(b) Flash MX - oversaturation of colors [25]

Figure 3.1 is not from the coral reef video. This one is used to describe the color saturation and temperature difference between the two codecs. As it can be seen Flash MX oversaturates color which results into the image looking a lot warmer. The saturated colors also give the image an impression of being sharper. According to the authors, the VP6 image on the left is much more true to the original.



(a)

(b)

Figure 3.2 (a) VP6 – Better quality picture (b) Flash MX - Blockiness of the subject and background [25]



(a)

(b)

Figure 3.3 (a) VP6 – Better picture quality (b) Flash MX - Loss of fine details in the background [25]



(a)

(b)

Figure 3.4 Blocking artifacts (a) VP6 – Better quality picture (b) Flash MX – Blocky artifacts can be visible in the subject and background [25]

The figures 3.2 thru 3.12 are from the coral reef exploration video. It can be observed in Figure 3.2 and figure 3.3 that the images from both codecs can maintain the details; but on careful examination it can be observed that a lot of the detail in the MX images is the blockiness from the 8x8 blocks rather than the details in the background. It can be observed in these images and the ones cited further in this section that a lot of detail in textured background regions is lost due the blockiness of the codec in MX. The blocking artifacts become significant and clear in the subject in the foreground in figure 3.2 and figure 3.4.



(a)

(b)

Figure 3.5 (a) VP6 (b) Flash MX - Artificial details can be observed [25]

The figure 3.5 is of Volitan Lionfish. There is a marked difference in how beautiful fish looks in both the images. In MX on the right observing the fins, many artificial details can be found from the blocking artifacts. Also some information seems to be jumbled up in the lower fins. Also the colors are significantly saturated. Apart from that, the careful observation of the coral background exhibits how the image on the left is lot truer than the MX image.



(a)

(b)

Figure 3.6 Low contrast backgrounds (a) VP6 – clear and sharp picture
(b) Flash MX – quality deteriorates[25]



(a)

(b)

Figure 3.7 Low contrast background (a) VP6 – Image details maintained (b) Flash MX – The reef in the background even loses its identity due to blockiness [25]

One more drawback that can be observed with MX is the inability to adapt to images with low and high contrast presence. Figures 3.6 and 3.7 are examples of that; the low contrast

ocean background affects the subject (the shark and the turtle). In the figure 3.7 the plight of the reef in the background is even worse. It almost loses its identity to the blocks.

Figures 3.8 thru 3.12 are examples with 2 pass encoding. Two-pass encoding allows the encoder to make better decisions about where to “spend” bits during compression, thereby improving the overall quality of the encoding.



(a)

(b)

Figure 3.8 Two pass encoding (a) VP6 – Better quality image (b) Flash MX [25]

The performance comparison is significant with 2-pass encoding. The image on the left in figure 3.8 is so well drawn it almost gives an impression of high quality content even at 150 kbps.



(a)

(b)

Figure 3.9 Low contrast background image (2 pass encoding) (a) VP6 (b) Flash MX [25]



(a)

(b)

Figure 3.10 Low contrast background (2 pass encoding) (a) VP6 – Better quality
(b) Flash MX – Blocky image [25]

Figures 3.9 and 3.10 are other examples of how poorly the old MX performs when there is a low contrast background and a high contrast foreground. The VP6 images appear so clean compared to the blocky background images on the right for both the examples.



(a)

(b)

Figure 3.11 (a) VP6 (b) Flash MX - Fish in background almost appear like artifacts of low contrast ocean background [25]

Figure 3.11 is another example where low contrast background artifacts and pseudo-sharpness in the MX image on the right leads to almost the loss of details of the small fishes. They appear more like motion artifacts in this figure.



(a)

(b)

Figure 3.12 Absolute loss of visual information (2 pass encoding)
 (a) VP6 – Quality maintained (b) Flash MX image [25]

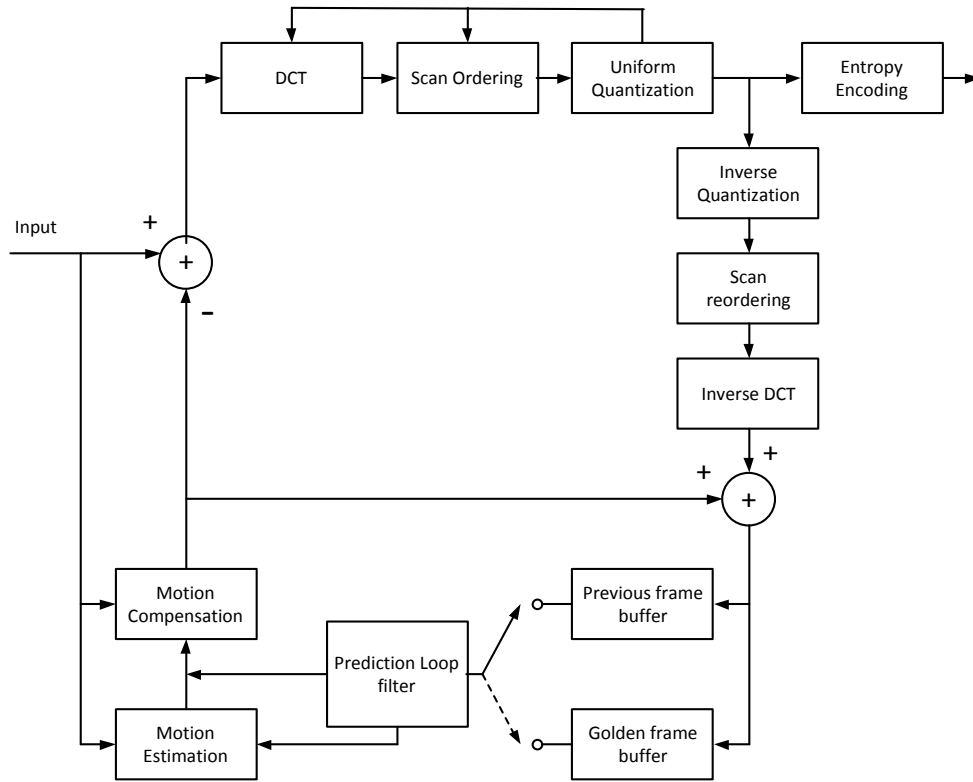
Finally figure 3.12 sums up the study by showing how difficult MX finds in low contrast scenarios compared to the performance of VP6. On the whole, it can be observed that MX performs poorly compared to VP6 for low contrast images, oversaturates colors and unnecessarily sharpens the images and also has blocky artifacts.

3.3 VP6 Algorithm Fundamentals

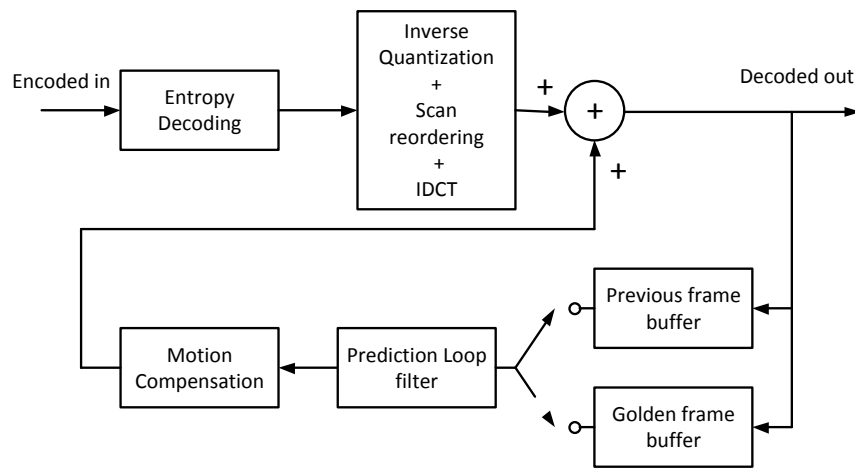
VP6 is a motion compensation and discrete cosine transform based codec like most of the open source codecs in the market [27]. Figure 3.13 shows approximate block diagrams of a VP6 encoder. Various coding tools and techniques in the block diagram are explained in sections 3.4 thru 3. 11. A high level overview of the codec fundamentals is as below [27].

- YUV 4:2:0 image format
- Macro-block (MB) based coding (MB is 16x16 luma plus two 8x8 chroma)
- $\frac{1}{4}$ pixel accuracy motion compensated prediction
- 8x8 DCT transform
- 64-level linear quantizer
- Prediction loop filter
- Frame variable quantization level
- Scaling on output after decode

- Two entropy coding strategies: Huffman & Binary Arithmetic (BoolCoder)
- Extensive context-based entropy coding strategy



(a)



(b)

Figure 3.13 VP6 – (a) Encoder block diagram
(b) Decoder block diagram

3.4 Coding profiles in VP6

Certain techniques used within the VP6 codec require significant computational resources that may not be available on low-end or even higher end processors for the very large image formats. So in order to distribute resources and tools in the codecs to justify the platform and the end user requirements two different profiles are defined in VP6 – VP6 Simple profile and Advanced profile [27].

Each frame header contains a flag, VpProfile, which indicates the profile that was used to code it. In both profiles the BoolCoder is used for encoding block and macro-block coding mode decisions and motion vectors in the first data partition.

When encoding in Simple Profile the DCT tokens are encoded in a second data partition, indicated in the bitstream by setting the MultiStream flag in the frame header. Furthermore, to reduce computational complexity both the prediction loop-filter and bi-cubic prediction filter are disabled.

When using Advanced Profile the second partition is optional depending on the MultiStream flag in the frame header. Where it is absent, all encoded data appears as single partition encoded using the BoolCoder. The second partition may be encoded using either the Huffman or BoolCoder entropy schemes. In addition, the use of the prediction loop-filter is optionally enabled, depending on a flag in the frame header, and the prediction filter type may be dynamically switched between bi-linear and bi-cubic variants.

In either profile where the second partition is present the UseHuffman flag in the frame header signifies whether the data is encoded using the Huffman or BoolCoder entropy schemes [27].

3.5 Types of Frames

VP6 defines only two frame types, intra-coded and inter-coded [27].

Intra, or I-frames, like any other codec do not use reference frames for reconstruction. As I-frames are the point where no previous decoding is required they are a method of fast

random access. The I-frames are not encoded with highly computationally involved intra-prediction as in H.264. The intra-coding technique used in VP6 is explained in section 3.6.

Inter prediction or P-frames, are encoded differentially with respect to a previously encoded reference frame in the sequence. Figure 3.14 describes previous frame prediction. This reference frame may either be the reconstruction of the immediately previous frame in the sequence or a stored previous frame known as the Golden Frame [27], described in section 3.5.1.

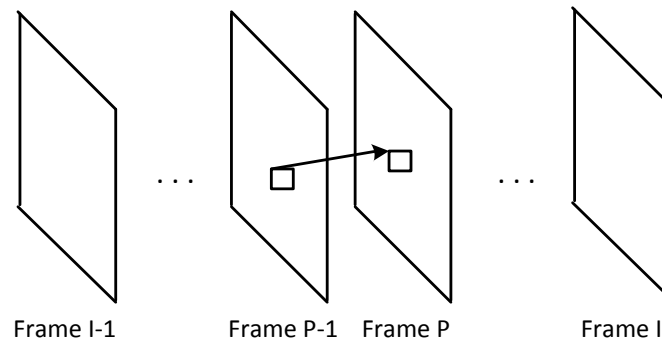
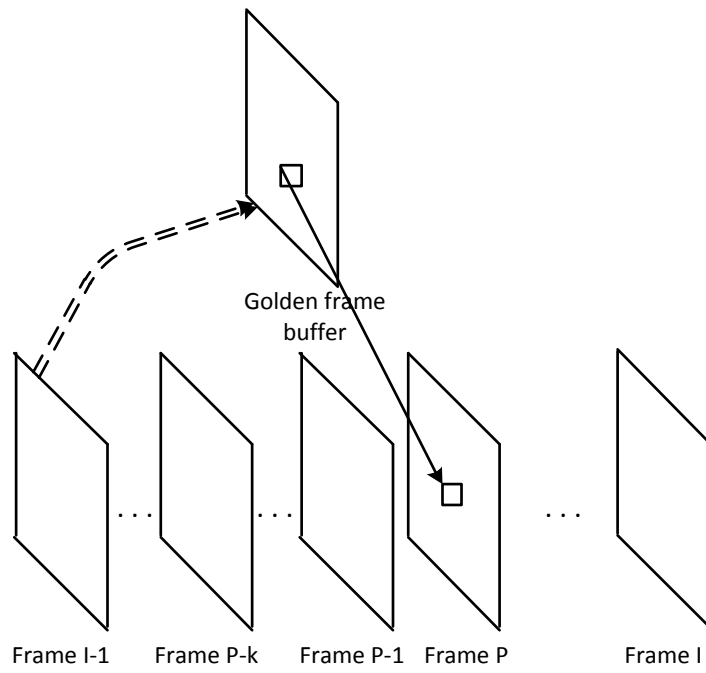


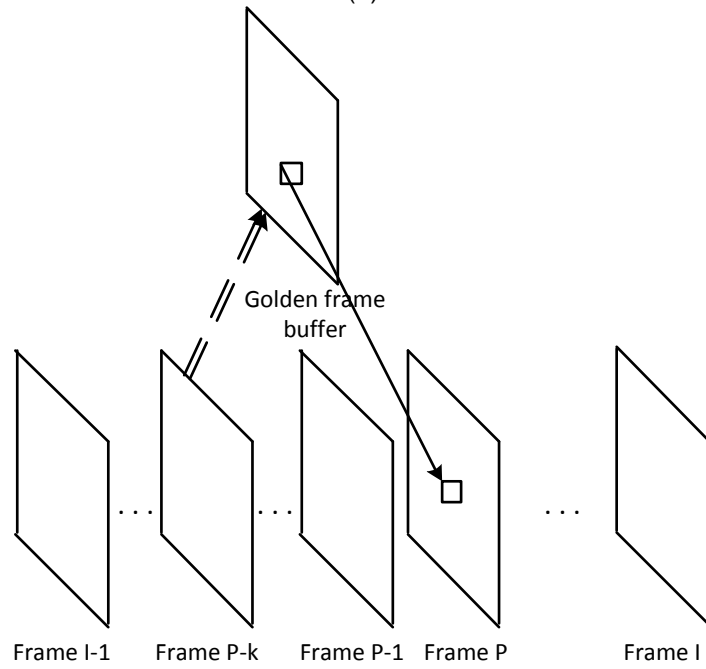
Figure 3.14 Previous frame prediction

3.5.1. Golden frames

The alternative prediction, or Golden Frame, is a frame buffer that by default holds the last decoded I-frame but it may be updated at any time. Figures 3.15 (a) and (b) show both the scenarios for golden frames. A flag in the frame header indicates to the decoder whether or not to update the Golden Frame buffer. To update the Golden frame the current frame is first decoded and then copied in its entirety into the Golden frame buffer.



(a)



(b)

Figure 3.15 Golden frame prediction (a) Golden frame buffer has the default i.e. last decoded I-frame (b) Golden frame buffer is updated

VP6 does not use backward or bi-directional prediction. So there are no B-frames as found in MPEG or H.264 [27].

3.6 MB Modes

VP6 has 10 Macroblock (MB) signalling modes - 1 intra-mode and 9 inter-modes [27].

3.6.1. MB Modes in I-frames (Intra-mode)

When the frame is an I frame, only intra-mode is used. So no signaling mode is required. Unlike H.264 no spatial prediction is used in CODE_INTRA (intra-mode in VP6). Each of its 6 blocks (4 Luma and 2 Chroma) is forward DCT encoded after the fixed value 128 is subtracted from each sample value; subtraction of 128 helps in improving DCT accuracy.

3.6.2. MB Modes in P-frames (Inter-modes and Intra-mode)

For P frames motion compensation is used. So Macroblocks are predicted using prediction frames. A prediction frame can be the previous frame or a golden frame. The motion vectors are specified in $\frac{1}{4}$ pixel units (i.e. $\frac{1}{4}$ sample precision for luma and $\frac{1}{8}$ sample precision for chroma).

The intra-mode used in P frames is exactly like the I-frames.

There are 9 other inter-modes defined. These modes depend on whether the Motion Vector (MV) is newly calculated or used from one of the neighboring MBs. The neighbor MBs used for prediction are classified as Near and Nearest blocks [27].

3.6.2.1 Nearest and Near blocks

In certain circumstances it is much more efficient to specify that a MB has the same MV as one of its nearest neighbors, rather than coding a new MV. For this reason VP6 defines the concept of the Nearest Motion Vector and Near Motion Vector. These are defined as first 2 non (0,0) MVs as encountered – the first being Nearest and second Near. The neighboring blocks and their order are described in the Fig 3.16. For the neighboring blocks to be labeled as Nearest or Near, they should be encoded using the same reference frame as the current MB. If no such block exists than Nearest and Near MVs are undefined.

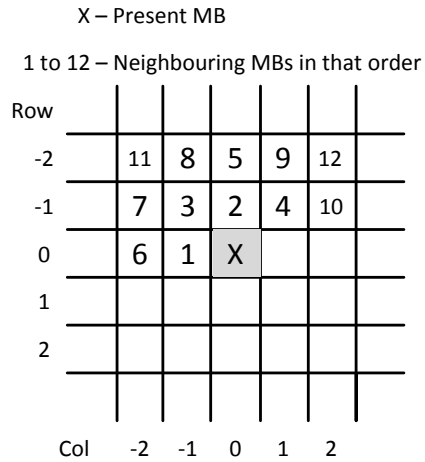


Fig 3.16 Order of the adjacent blocks to find Near and Nearest neighbors

The different coding modes are described in Table 3.1. Most of the coding modes are self-explanatory.

Table 3.1 MB coding modes in VP6 [27]

Coding mode	Prediction frame	Motion vector (MV)
CODE_INTER_NO_MV	Previous frame reconstruction	Fixed: (0,0)
CODE_INTRA	None	None
CODE_INTER_PLUS_MV	Previous frame reconstruction	Newly calculated MV
CODE_INTER_NEAREST_MV	Previous frame reconstruction	Same MV as Nearest block
CODE_INTER_NEAR_MV	Previous frame reconstruction	Same MV as Near block
CODE_USING_GOLDEN	Golden frame	Fixed: (0,0)
CODE_GOLDEN_MV	Golden frame	Newly calculated MV
CODE_INTER_FOURMV	Previous frame reconstruction	Each of the four luma-blocks has associated MV
CODE_GOLD_NEAREST_MV	Golden frame	Same MV as Nearest block
CODE_GOLD_NEAR_MV	Golden frame	Same MV as Near block

For previous frame reconstruction, it is also possible to have a different motion vector for all the blocks in the MB. In this coding mode CODE_INTER_FOURMV each of the four Y-blocks will be coded independently, each having an associated coding mode from a reduced set that excludes intra or any of the Golden Frame modes. The motion vector for the two chroma blocks is computed by averaging the four Y vectors (rounding away from zero) [27].

3.7 Motion Vectors

VP6 has 9 MB modes which involve motion prediction. The motion vectors are either explicitly calculated or the MV of a neighbor is used. 3 modes involve calculation of explicitly new motion vectors [27].

- `CODE_INTER_PLUS_MV`: A new motion vector is coded with reference to the previous frame reconstruction.
- `CODE_GOLDEN_MV`: A new motion vector is coded with reference to the Golden frame reconstruction
- `CODE_INTER_FOURMV`: A different mode may to be specified for each of the luma blocks from a subset of those available at the MacroBlock level (see Table 3.1). Each block coded with mode `CODE_INTER_PLUS_MV` will have its own explicitly coded motion vector.

If the Nearest MacroBlock exists and is either immediately to the left of (neighbor 1) or immediately above (neighbor 2) the current MacroBlock (as described in figure 3.13), than the new motion vectors are coded differentially with respect to the motion vector of the nearest MacroBlock. If such a block does not exist, the new MVs are coded absolutely.

The implicit motion vectors can have a case of no MV i.e. (0,0) or the motion vector from the nearest MB or the motion vector from the near MB. These modes are as below:

- `CODE_INTER_NO_MV`: Use the motion vector (0,0) applied to the previous frame reconstruction.
- `CODE_INTER_NEAREST_MV`: Use the motion vector from a previously coded nearest MacroBlock applied to the previous frame reconstruction.
- `CODE_INTER_NEAR_MV`: Use the motion vector from a previously coded near MacroBlock applied to the previous frame reconstruction
- `CODE_USING_GOLDEN`: Use the motion vector (0,0) applied to the Golden frame reconstruction.

- `CODE_GOLD_NEAREST_MV`: Use the motion vector from a previously coded nearest MacroBlock applied to the Golden frame reconstruction.
- `CODE_GOLD_NEAR_MV`: Use the motion vector from a previously coded near MacroBlock applied to the Golden frame reconstruction.

Nearest and Near MacroBlocks are explained in section 3.6.2.1.

3.7.1 Encoding

The motion vector values will have an x-component and y-component. Each of them can be categorized as either a short vector or a long vector. So a motion vector can have the x-component encoded as a short vector and the y-component a long vector or any such combination. The length is the length of the individual component (x component of y component); it is not the total magnitude.

- A short vector is defined as a vector with a length that is less than 8 in $\frac{1}{4}$ pixel units.
- A long vector is defined as a vector with a length that is greater than or equal to 8 and less than or equal to 127 in $\frac{1}{4}$ pixel units.

3.7.2 Prediction loop filtering

In order to create a prediction block for the non-zero motion vectors VP6 has a prediction loop filter. As it does not have traditional loop filtering, this filtering also helps in reducing the blocking artifacts. The Prediction Loop filter due to its dual usage is explained in section 3.8.

3.7.3 Filtering For fractional pixel motion compensation

VP6 supports the use of fractional pixel motion compensation up to $\frac{1}{4}$ sample precision for Luma and $\frac{1}{8}$ sample precision for chroma. Interpolation is used to determine sample values at non whole-pixel locations.

Two type of interpolation filtering is supported:

- Bilinear filtering: Using 2 tap filters (see Section 3.7.3.1).
- Bicubic filtering: Using 4 tap filters (see Section 3.7.3.2).

In “Simple Profile” Bicubic filtering is not allowed, so Bilinear filtering is used in all cases where fraction pixel predictors are required.

3.7.3.1 Bilinear filtering

The 1-D filter taps described in table 3.2 are used for bilinear filtering to ¼ sample precision in luma.

Table 3.2. Bilinear (1-D) filter taps for ¼ sample precision Luma filtering [27]

```
BilinearLumaFilters[4][2] =  
{  
  { 128, 0 }, // Full sample aligned  
  { 96, 32 }, // 1/4  
  { 64, 64 }, // 1/2  
  { 32, 96 }, // 3/4  
}
```

The 1-D filter taps described in table 3.3 are used for bilinear filtering to 1/8 sample precision in chroma.

Table 3.3. Bilinear (1-D) filter taps for 1/8 sample precision chroma filtering [27]

```
BilinearChromaFilters[8][2] =  
{  
  { 128, 0 }, // Full sample aligned  
  { 112, 16 }, // 1/8  
  { 96, 32 }, // 1/4  
  { 80, 48 }, // 3/8  
  { 64, 64 }, // 1/2  
  { 48, 80 }, // 5/8  
  { 32, 96 }, // 3/4  
  { 16, 112 } // 7/8  
}
```

In cases where the motion vector has a fractional component in both x and y direction an intermediate result is calculated by applying the filter in the x direction (horizontally). This intermediate result used as input to a second pass which filters in the y direction (vertically) to produce the final 2-d filtered output.

3.7.3.2 Bicubic filtering

Bicubic filter taps are calculated for 16 values of alpha from -0.25 to -1.00. For each value of alpha, there are 8 sets of coefficients corresponding to 1/8 pel offsets from 0 to 7/8. These values are only used in VP6.2 bitstreams. The filter tap values are described in table 3.4. The 17th entry in the table is used for VP6.1 bitstreams [27].

Table 3.4. Bicubic (4-tap) filter tabs for 1/8 pixel interpolation [27]

```
BicubicFilterSet[17][8][4] =
{ { 0, 128, 0, 0 }, // Full sample aligned, A ~= -0.25
  { -3, 122, 9, 0 }, // 1/8
  { -4, 109, 24, -1 }, // 1/4
  { -5, 91, 45, -3 }, // 3/8
  { -4, 68, 68, -4 }, // 1/2
  { -3, 45, 91, -5 }, // 5/8
  { -1, 24, 109, -4 }, // 3/4
  { 0, 9, 122, -3 }, // 7/8
},
{ { 0, 128, 0, 0 }, // A ~= -0.30
  { -4, 124, 9, -1 },
  { -5, 110, 25, -2 },
  { -6, 91, 46, -3 },
  { -5, 69, 69, -5 },
  { -3, 46, 91, -6 },
  { -2, 25, 110, -5 },
  { -1, 9, 124, -4 },
},
{ { 0, 128, 0, 0 }, // A ~= -0.35
  { -4, 123, 10, -1 },
  { -6, 110, 26, -2 },
  { -7, 92, 47, -4 },
  { -6, 70, 70, -6 },
  { -4, 47, 92, -7 },
  { -2, 26, 110, -6 },
  { -1, 10, 123, -4 },
},
},
```

Table 3.4 - *Continued*

```

{ { 0, 128, 0, 0 }, // A ~= -0.40
{ -5, 124, 10, -1 },
{ -7, 110, 27, -2 },
{ -7, 91, 48, -4 },
{ -6, 70, 70, -6 },
{ -4, 48, 92, -8 },
{ -2, 27, 110, -7 },
{ -1, 10, 124, -5 },
},
{ { 0, 128, 0, 0 }, // A ~= -0.45
{ -6, 124, 11, -1 },
{ -8, 111, 28, -3 },
{ -8, 92, 49, -5 },
{ -7, 71, 71, -7 },
{ -5, 49, 92, -8 },
{ -3, 28, 111, -8 },
{ -1, 11, 124, -6 },
},
{ { 0, 128, 0, 0 }, // A ~= -0.50
{ -6, 123, 12, -1 },
{ -9, 111, 29, -3 },
{ -9, 93, 50, -6 },
{ -8, 72, 72, -8 },
{ -6, 50, 93, -9 },
{ -3, 29, 111, -9 },
{ -1, 12, 123, -6 },
},
{ { 0, 128, 0, 0 }, // A ~= -0.55
{ -7, 124, 12, -1 },
{-10, 111, 30, -3 },
{-10, 93, 51, -6 },
{ -9, 73, 73, -9 },
{ -6, 51, 93, -10 },
{ -3, 30, 111, -10 },
{ -1, 12, 124, -7 },
},

```

Table 3.4 - *Continued*

```

{ { 0, 128, 0, 0 }, // A ~= -0.60
{ -7, 123, 13, -1 },
{-11, 112, 31, -4 },
{-11, 94, 52, -7 },
{-10, 74, 74, -10 },
{ -7, 52, 94, -11 },
{ -4, 31, 112, -11 },
{ -1, 13, 123, -7 },
},
{ { 0, 128, 0, 0 }, // A ~= -0.65
{ -8, 124, 13, -1 },
{-12, 112, 32, -4 },
{-12, 94, 53, -7 },
{-10, 74, 74, -10 },
{ -7, 53, 94, -12 },
{ -4, 32, 112, -12 },
{ -1, 13, 124, -8 },
},
{ { 0, 128, 0, 0 }, // A ~= -0.70
{ -9, 124, 14, -1 },
{-13, 112, 33, -4 },
{-13, 95, 54, -8 },
{-11, 75, 75, -11 },
{ -8, 54, 95, -13 },
{ -4, 33, 112, -13 },
{ -1, 14, 124, -9 },
},
{ { 0, 128, 0, 0 }, // A ~= -0.75
{ -9, 123, 15, -1 },
{-14, 113, 34, -5 },
{-14, 95, 55, -8 },
{-12, 76, 76, -12 },
{ -8, 55, 95, -14 },
{ -5, 34, 112, -13 },
{ -1, 15, 123, -9 },
},

```

Table 3.4 - *Continued*

```

{ { 0, 128, 0, 0 }, // A ~= -0.80
{-10, 124, 15, -1 },
{-14, 113, 34, -5 },
{-15, 96, 56, -9 },
{-13, 77, 77, -13 },
{ -9, 56, 96, -15 },
{ -5, 34, 113, -14 },
{ -1, 15, 124, -10 },
},
{ { 0, 128, 0, 0 }, // A ~= -0.85
{-10, 123, 16, -1 },
{-15, 113, 35, -5 },
{-16, 98, 56, -10 },
{-14, 78, 78, -14 },
{-10, 56, 98, -16 },
{ -5, 35, 113, -15 },
{ -1, 16, 123, -10 },
},
{ { 0, 128, 0, 0 }, // A ~= -0.90
{-11, 124, 17, -2 },
{-16, 113, 36, -5 },
{-17, 98, 57, -10 },
{-14, 78, 78, -14 },
{-10, 57, 98, -17 },
{ -5, 36, 113, -16 },
{ -2, 17, 124, -11 },
},
{ { 0, 128, 0, 0 }, // A ~= -0.95
{-12, 125, 17, -2 },
{-17, 114, 37, -6 },
{-18, 99, 58, -11 },
{-15, 79, 79, -15 },
{-11, 58, 99, -18 },
{ -6, 37, 114, -17 },
{ -2, 17, 125, -12 },
},

```

Table 3.4 - *Continued*

```

{ { 0, 128, 0, 0 }, // A ~= -1.00
{-12, 124, 18, -2 },
{-18, 114, 38, -6 },
{-19, 99, 59, -11 },
{-16, 80, 80, -16 },
{-11, 59, 99, -19 },
{-6, 38, 114, -18 },
{-2, 18, 124, -12 },
},
{
{ 0, 128, 0, 0 }, // Coefficients for VP6.1 bitstreams
{-4, 118, 16, -2 },
{-7, 106, 34, -5 },
{-8, 90, 53, -7 },
{-8, 72, 72, -8 },
{-7, 53, 90, -8 },
{-5, 34, 106, -7 },
{-2, 16, 118, -4 }
}
}

```

3.7.4 Support for unrestricted motion vectors

VP6 supports the concept of unrestricted motion vectors (UMV). This means that it is legal for a motion vector to point to a prediction block that extends beyond the borders of the image. To support this feature and also the playback scaling features of the codec the reconstruction buffers are extended by 48 sample points in all directions as described in figure 3.17.

The buffers are extended by duplicating the edge values 48 times. This is done first in x (horizontally) and then in the y (vertically).

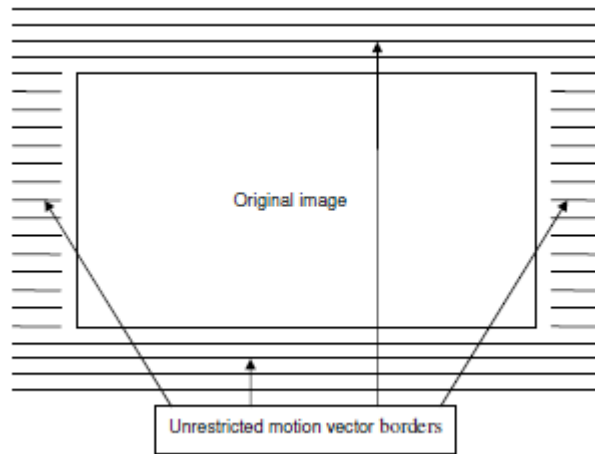


Figure 3.17 Support for motion vector beyond the image boundaries [27]

3.8 Prediction Loop Filtering

VP6 does not have a traditional reconstruction buffer loop filter; but it supports filtering of pixels adjacent to 8x8 block boundaries in the prediction frame (previous frame or golden frame reconstruction as appropriate), as part of the process for creating a prediction block for non-zero motion vectors. As with traditional loop filters this helps to reduce blocking artifacts, but the filtering is not carried out in place within the reconstruction buffer. Rather, the output is copied into a separate temporary buffer. This is done before any filtering required for fractional pixel motion compensation (see Section 3.7.3) [27].

The prediction Loop filter is disabled in Simple Profile. In other profiles it is enabled if the UseLoopFilter flag in the frame header is set to 1.

If the prediction block defined by a motion vector straddles an 8x8 block boundary in the prediction frame then a de-blocking and/or de-ringing filter is applied to the pixels adjacent to the boundary to reduce any discontinuities (see Figure 3.17).

Two filter options are as follows:

- A deringing filter: has de-blocking and de-ringing characteristics

- A deblocking filter: has only de-blocking characteristic. The deblocking loop filter comprises a 4-tap filter (1, -3, 3, -1) and a quantizer dependant bounding function applied across the horizontal and vertical block boundaries. The prediction loop filter coefficients are described in table 3.5 [27]

Table 3.5 Prediction loop filter limit values [27]

```
PredictionLoopFilterLimitValues [64] =
{
30, 25, 20, 20, 15, 15, 14, 14,
13, 13, 12, 12, 11, 11, 10, 10,
9, 9, 8, 8, 7, 7, 7, 7,
6, 6, 6, 6, 5, 5, 5, 5,
4, 4, 4, 4, 3, 3, 3, 3,
2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2,
1, 1, 1, 1, 1, 1, 1, 1
}
```

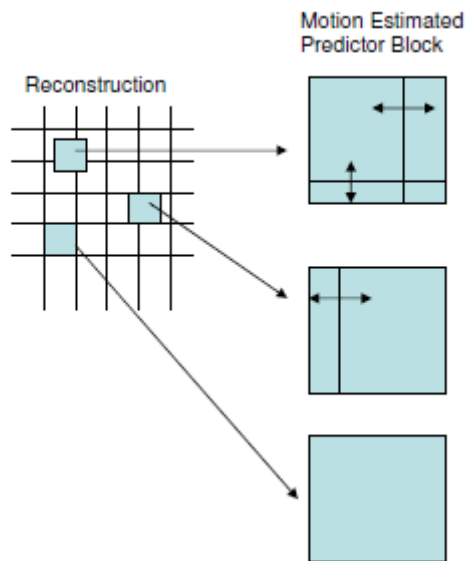


Figure 3.18 Prediction loop filtering of 8x8 block boundaries [27]

3.9 DCT, scan orders and coefficient token set

In order to reduce the complexity at the decoder, VP6 uses a slightly modified non-standard fixed point integer inverse discrete cosine transform (DCT); the DCT has 14 bits of precision is used to convert the coefficients back to pixels or pixel difference values. This transform is based upon the paper by M. Vetterli, A. Ligtenberg "A Discrete Fourier-Cosine Transform Chip" IEEE Journal on Selected Areas of Communications, Vol. SAC-4, pp 49-61, Jan. 1986, pp 49-61 [28]. The authors in this paper propose a DCT technique with focus on variable complexity algorithms (VCAs) that can adjust the forward DCT complexity as a function of target quantization to be used. This can provide faster performance when quantization is coarser. Computations needed to generate zero or small-magnitude coefficients can be safely omitted if the locations of those coefficients are known. This also enables straightforward classification of blocks of transformed and quantized data based on the location of zero coefficients for inverse DCT cases. The grouping of zero coefficients enables us to have an IDCT algorithm with reduced complexity.

Here the forward DCT needs to be able to predict the sparseness of the quantized DCT output accurately and with minimal complexity overhead. This has to be done before transform and quantization are applied. So the algorithm takes into consideration the quantization levels and the input block characteristics [28].

In an attempt to be able to group the non-zero coefficients together at the beginning of the group, customized scanning order of DCT coefficients is possible (section 3.9.1.2).

3.9.1 Scan orders

Scan reordering is the process of providing customized scanning order. If we number the 64 coefficients of the 8x8 transformed block in raster order such that coefficients 0 and 63 are the DC and highest order AC coefficients, respectively, then the scan re-ordering is specified by a 64 element array which gives the new ordering. The coefficients appear in the

modified order in the bitstream. The decoder rearranges them back to raster order before inverse quantization and IDCT [27].

3.9.1.1 Default scan order

The default scan order is the standard zig-zag order shown in Figure 3.19.

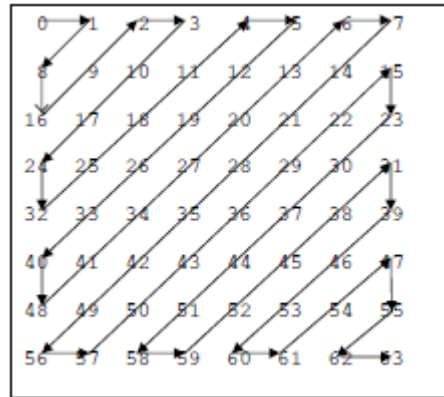


Figure 3.19 Default zig-zag scan order [27]

3.9.1.2 Custom scan order

Per frame custom scan orders are supported in VP6. The encoder signals the use of customized scanning orders.

For intra-coded frames the scan order is first set to the appropriate default. This default is then updated using delta information encoded in the bitstream. For inter-coded frames deltas are applied to the custom scan order used in the previous frame rather than to the one of the default scan orders.

In all scan orders the first DCT coefficient is always the DC coefficient.

To specify a custom scan order, each AC coefficient (in zig zag order) is assigned to one of the bands as mentioned in table 3.7. The table describes the 16 bands into which the 63 AC positions in modified scan order are split. Within each band the coefficients are then sorted into ascending order based upon the original zig-zag scan order. The decoder maintains a table

3.6 for decoding the coefficient band update information for each of the AC coefficients in standard zigzag order [27].

Table 3.6 Coefficient band update probabilities [27]

```
CoeffBandUpdateFlagProbs[64] =  
{  
  NA, 132, 132, 159, 153, 151, 161, 170,  
  164, 162, 136, 110, 103, 114, 129, 118,  
  124, 125, 132, 136, 114, 110, 142, 135,  
  134, 123, 143, 126, 153, 183, 166, 161,  
  171, 180, 179, 164, 203, 218, 225, 217,  
  215, 206, 203, 217, 229, 241, 248, 243,  
  253, 255, 253, 255, 255, 255, 255, 255,  
  255, 255, 255, 255, 255, 255, 255, 255  
}
```

3.9.2 DCT encoding and coefficient token set

The DCT involved encoding at 3 levels - predictive encoding of the DC coefficients, encoding the AC coefficients and encoding the zero-runs of DC and AC coefficients [27].

3.9.2.1 DC prediction

The DC coefficient for a block is reconstructed at the decoder by adding together a prediction value and a prediction error. The prediction error is encoded in the bitstream and decoded. The prediction value is computed from the DC values of neighboring blocks in the current frame that have already been decoded.

For a particular block the DC values of up to two particular immediate neighbors contribute to the prediction. As shown figure 3.20 the two blocks concerned are the blocks immediately to the left of and immediately above the current block [27].

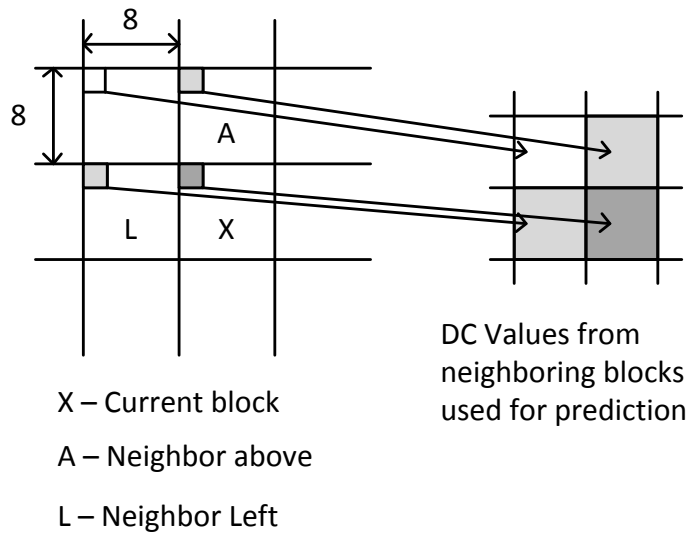


Figure 3.20 DC prediction using neighbors

The DC value of a neighboring block only contributes to the prediction of the DC value for a particular block if all of the following conditions are satisfied:

- The neighboring block exists; there is no left neighbor for blocks at the left edge and no above neighbor for blocks at the top edge of the frame,
- The neighboring block was predicted from the same reference frame as the block being predicted (last frame reconstruction or golden frame),
- Inter-coded blocks can only be predicted by neighboring inter-coded blocks and Intra-coded blocks can only be predicted by neighboring intra-coded blocks [27].

The table 3.7 describes how the predicted values are calculated based on the whether the neighboring blocks are present or one or none of the neighboring blocks are present.

Table 3.7 DC prediction based on presence of neighboring blocks [27]

Left (L) available	Above (A) available	Predictor
NO	NO	Last decoded DC value for a block with the same prediction frame
NO	YES	A
YES	NO	L
YES	YES	$(L + A + \text{sign}(L + A)) / 2$

3.9.2.2 Coefficient token set

Following set of 12 tokens described in table 3.8 is used to represent the quantized DCT coefficients.

Table 3.8 DCT token set and extra bits [27]

Index	Token	Min	Max	Extra bits (including sign)	Arithmetic encoding of the extra bits
0	ZERO_TOKEN	0	0	*	
1	ONE_TOKEN	1	1	1	B(128)
2	TWO_TOKEN	2	2	1	B(128)
3	THREE_TOKEN	3	3	1	B(128)
4	FOUR_TOKEN	4	4	1	B(128)
5	DCT_VAL_CATEGORY1	5	6	2	B(159), B(128)
6	DCT_VAL_CATEGORY2	7	10	3	B(165), B(145), B(128)
7	DCT_VAL_CATEGORY3	11	18	4	B(173), B(148), B(140), B(128)
8	DCT_VAL_CATEGORY4	19	34	5	B(176), B(155), B(140), B(135), B(128)
9	DCT_VAL_CATEGORY5	35	66	6	B(180), B(157), B(141), B(134), B(130), B(128)
10	DCT_VAL_CATEGORY6	67	2114	12	B(254), B(254), B(243), B(230), B(196), B(157), B(153), B(140), B(133), B(129), B(128)
11	DCT_EOB_TOKEN	N/A	N/A	**	

Min-value in the table 3.8 represents the smallest value that can be encoded using that token. The extra-bits reflect the range of values for that token. The MSB of the magnitude is sent first whereas the last extra-bit is always the sign bit. In the arithmetic encoding the extra bits are each encoded with differing probabilities. In Huffman encodings these bits are just pumped on to the bitstream.

Probability values and contextual information are used to encode the DCT coefficients into these tokens. These probability values are stored in tables that are kept by the decoder and may be updated on a frame by frame basis.

At the decoder a Binary Coding Tree for DC and AC Tokens as shown in fig 3.21 is specified for decoding DCT coefficient tokens. The bitstream provides the set of probabilities for taking the 0 branch at each node in the tree. Thus the bitstream is encoded with probabilities to

take the zero branch at each node of the binary tree which is used by the decoder to decode the DCT tokens. The same set of probabilities can be converted to a set of Huffman probabilities using an algorithm available to the decoder [27].

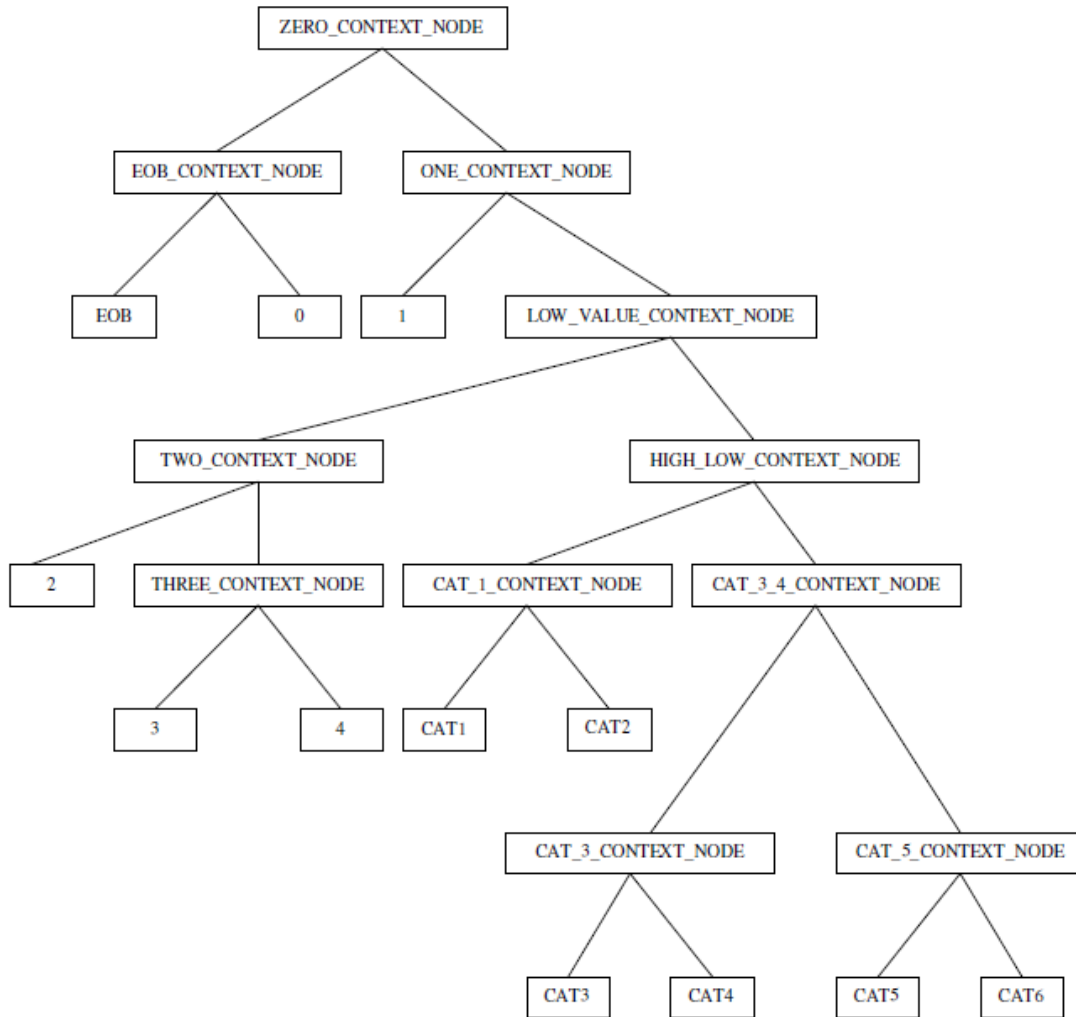


Figure 3.21 Binary coding tree for AC and DC contexts [27]

3.9.2.3 DC decoding

For DC, the decoder maintains two sets (of length 11 with a value for each of the 11 tokens) of probabilities each for the Y plane and the UV planes. These probabilities get updated on frame by frame basis; the bitstream provides the information to update these probabilities at the decoder [27].

Arithmetic and Huffman decoding of the DC Coefficients

An arithmetically encoded DC value is decoded using the updated probabilities. Decoding the DC values makes use of the contextual information regarding whether the blocks immediately to the left of and above the current block have 0 or non 0 dc values (table 3.9).

If Huffman coding of the DC tokens has been used, an algorithm available at the decoder is used to produce the Huffman decoding tree directly from the BoolCoder tree. These probabilities are then used to decode the Huffman encoded DC values.

Table 3.9 DC Node contexts [27]

Index	Situation
0	Left block's predicted DC was 0 and above block's predicted DC was 0z
1	Either left block's predicted DC value is non-zero or above block's predicted value is non-zero but not both
2	Both left block's predicted and above block's predicted DCs

3.9.2.4 AC decoding

To decode ac coefficients the decoder maintains a 4 dimensional set of probabilities. The set of 11 probabilities for the tokens is maintained for each of Y or UV plane and 6 bands of coefficients as mentioned in the table 3.10. The probability set is different for each of the context situation described in table 3.11. The bitstream provides the information to update these probabilities at the decoder [27].

Table 3.10 AC coefficient bands [27]

Index	Situation
0	Coefficient 1
1	Coefficients 2-4
2	Coefficients 5-10
3	Coefficients 11-21
4	Coefficients 22-36
5	Coefficients 37-63

Table 3.11 AC preceding decoded coefficient context [27]

Index	Situation
0	Preceding decoded coefficient (in current scan order) for the current block was 0
1	Preceding decoded coefficient (in current scan order) for the current block was 1
2	Preceding decoded coefficient (in current scan order) for the current block was greater than 1

Arithmetic and Huffman Decoding of the AC Coefficients

An arithmetically encoded AC value is decoded using the updated probabilities at the decoder from the bitstream. All 4 sets of contextual information - the Y or UV plane, the band to which the coefficient belongs to, whether the preceding coefficient in the block was 0, 1 or > 1 and the corresponding token are required for decoding the AC value.

Decoding Huffman encoded AC coefficients requires the use of 24 separate Huffman trees stored in a 3 dimensional array. There is a different tree depending on which plane (Y or UV) the coefficient belongs to, whether the preceding coefficient in the block was 0, 1 or > 1 and to which band as described in table 3.12 the coefficient falls in [27].

Table 3.12 AC coefficient bands for Huffman [27]

Index	Situation
0	Coefficient 1
1	Coefficients 2-4
2	Coefficients 5-10
3	Coefficients 11-63

3.9.2.5 Decoding zero runs

To decode zero runs the decoder must maintain a 2 dimensional set of probabilities. The first dimension of the ZeroRun probabilities is indexed by the band that the zero coefficient starts as described in table 3.13. The second dimension of the probability table is indexed depending on run-length value as described in table 3.14 [27].

Table 3.13 Zero runs coefficient bands [27]

Index	Situation
0	Coefficients 1-5
1	Coefficients 6-63

Table 3.14 Zero runs node index [27]

Index	Run length
0	Probability of Run length > 4
1	Probability of Run length > 2
2	Probability of Run length > 1
3	Probability of Run length > 3
4	Probability of Run length > 8
5	Probability of Run length > 6
6	Probability of Run length > 5
7	Probability of Run length > 7
8	Probability of bit (Run length – 9) & 1
9	Probability of bit ((Run length – 9) >> 1) & 1
10	Probability of bit ((Run length – 9) >> 2) & 1
11	Probability of bit ((Run length – 9) >> 3) & 1
12	Probability of bit ((Run length – 9) >> 4) & 1
13	Probability of bit ((Run length – 9) >> 5) & 1

3.10 Quantization

Each motion predicted 8x8 block of a video frame is transformed by the encoder to a set of 64 coefficients via the discrete cosine transform. These 64 coefficients are then quantized by means of 2 separate uniform scalar quantizers: 1 for the DC coefficient, and 1 for all 63 of the AC coefficients.

Reversing the uniform scalar quantizer involves performing integer multiplication on each of its 64 coefficients. The quantization value (multiplicand) for DC is determined by indexing the table DcQuantizationTable (table 3.15). Likewise the Ac quantization value is determined by indexing the table AcQuantization Table (table 3.15) [27].

Table 3.15 DC and AC Quantization values [27]

```
DcQuantizationTable[ 64 ] =  
{  
188, 188, 188, 188, 180, 172, 172, 172,  
172, 172, 168, 164, 164, 160, 160, 160,  
160, 140, 140, 140, 140, 132, 132, 132,  
132, 128, 128, 128, 108, 108, 104, 104,  
100, 100, 96, 96, 92, 92, 76, 76,  
76, 76, 72, 72, 68, 64, 64, 64,  
64, 64, 60, 44, 44, 44, 40, 40,  
36, 32, 28, 20, 12, 12, 8, 8  
}  
ACQuantizationTable[64] =  
{  
376, 368, 360, 352, 344, 328, 312, 296,  
280, 264, 248, 232, 216, 212, 208, 204,  
200, 196, 192, 188, 184, 180, 176, 172,  
168, 160, 156, 148, 144, 140, 136, 132,  
128, 124, 120, 116, 112, 108, 104, 100,  
96, 92, 88, 84, 80, 76, 72, 68,  
64, 60, 56, 52, 48, 44, 40, 36,  
32, 28, 24, 20, 16, 12, 8, 4  
}
```

3.11 Entropy Coding

There are two different entropy coding strategies for encoding of the DCT coefficient tokens in VP6 – Huffman coder and BoolCoder [27].

The Huffman coder is a very computationally efficient method that is well suited to speed optimization and has reasonable compression performance. It is typically used in very high data-rate scenarios on low to mid-range processors because it can handle the large volume of tokens more efficiently than the BoolCoder.

The BoolCoder is a simplified binary arithmetic coder allowing tokens to be encoded with fractions of a bit. It is much more efficient in terms of compression performance than the Huffman coder, but this comes with a significantly increased computational complexity.

Both the Huffman coder and BoolCoder use binary decision trees to represent multi-bit syntax elements. In each case the tree is traversed as a sequence of branch decisions is read from the bitstream until a leaf node is reached. Each leaf node has an associated syntax element.

The difference between the two schemes lies in the way the branching decisions are encoded at the tree nodes. The Huffman coder uses an entire bit to encode the branching decision at a given node. The BoolCoder makes use of a probability value called as the node probability. The node probability is the probability to branch left (zero) at a given node. So the BoolCoder can achieve sub-bit decision costs [27].

Whereas the Huffman coder is completely specified by the binary decision tree, the BoolCoder additionally requires the definition of a set of Node Probabilities. Node probabilities are specified as an array of values, specified in order as the tree is traversed in depth-first order. Node probabilities are represented on a linear 8-bit scale: 0 represents probability 0, 255 represents probability 1. However, the value 0 is explicitly forbidden, so the valid range is as follows:

$$1 \leq \text{Node Probability} \leq 255 \quad (3.1) [27]$$

3.11.1 Use of context information

A lot of statistical correlation exists not only between adjacent symbols and but the various coding parameters and tools used for encoding neighboring blocks. For example if the nearest block is encoding with CODE_INTRA there is about 85% chance that the present block may be encoded using CODE_INTRA and as low as 3% chance that the current block may be encoded using CODE_INTER_PLUS_MV. So the coding mode of the near / nearest blocks can be used as context information for encoding the current block coding mode. Such use of contexts which exploit the correlation between the coding parameters of adjacent blocks and pixels is a very useful way of reducing the amount of statistical information in the bitstream. By using information already available at the decoder weighting may be applied to a set of baseline

probabilities to adapt them better to the current coding environment. This results in more efficient entropy coding.

So a conditional probability distribution, derived from a baseline distribution with respect to a defined context is used for efficient entropy coding [27].

3.11.2 Huffman coder

In order to decode a syntax element encoded with Huffman encoder the Huffman decoder traverses a specified binary tree, at each node branching to either the left or right child-node as dictated by the next bit read from the bitstream (0 indicates left, 1 indicates right). Traversal stops when a leaf node is encountered; each leaf node corresponds to a particular syntax element.

The Huffman tree is the standard Huffman tree. This tree is constructed using the set of leaf node probabilities. However instead of encoding the leaf node probabilities, VP6 encodes a set of node probabilities to be compatible to the way the BoolCoder trees are encoded.

So the decoder needs to translate the node probabilities available from the bitstream to a set of leaf node probabilities so that they can be used to create the Huffman tree. The decoding process that follows is the process of traversing this tree with appropriate branch decisions. The leaf-node probability is calculated as the product of the individual node probabilities as the tree is traversed from its root to the leaf node, with appropriate normalization [27].

3.11.3 BoolCoder

The BoolCoder is based on the same principles as a binary arithmetic coder. It codes successive 0 or 1 decisions by continuously sub-dividing an initial unit interval in the ratio of the relative probabilities that a 0 and 1 will occur. Encoding multi-bit entities can be considered as traversing a binary decision tree where at each node there is an associated probability of taking the left, or zero, branch. This probability is referred to as the Node Probability. The probability of taking a right, or 1, branch is therefore one minus the node probability [27].

CHAPTER 4

COMPARISON BETWEEN VP6 AND H.264 STANDARDS

4.1 Introduction

Before Adobe adopted H.264 video coding standard [5] for flash, VP6 was the only format that existed on flash video. With the kind of penetration [7] that flash has, VP6 formed a huge part of the video-on-the-web ecosystem. Considering such outreach of flash video, curiosity on its comparison with the new flash standard - H.264 existed. A number of references [34] thru [37] on the comparison of the two codecs from the user perspective, especially on the content creation aspect are available on the web. The adoption of H.264 which is a very widely popular video coding standard should not affect the already existing VP6 standard. Both the codecs have their own advantages and there is a very high probability of coexistence on the flash platform and the streaming video market [35].

It is very important to be able to understand the differences between the two encoding standards before the process of transcoding between the two standards is described. This chapter describes the comparison of performance between the two codecs for comparable profiles. The two profiles looked at are VP6 Simple profile and H.264 Baseline profile. The existence of both the standards on the flash ecosystem affects the online video streaming and broadcast market the most. So a comparison from the content creator and end-user perspective is also described based on the references mentioned earlier [34] [35] [36].

4.2 Comparison of features and coding tools

H.264 is a set of encoding tools to provide high quality video at low bitrates. A lot of encoding tools employed in order to achieve that include significant computation to reduce the bitrate. So H.264 is significantly computationally involved compared to other codecs. The main

advantage that VP6 standard on the other hand offers is that it can keep the encoding process as simple as possible, thereby achieving processor friendliness. VP6 stresses the processor significantly less compared to H.264. As a result there is a lot of difference between different encoding tools employed in the two codecs. Before the adoption of VP6, flash video used the Sorenson Spark codec which was based on the H.263 video standard [25]. The table 4.1 provides a high level overview of the difference between the various features of all three flash video standards – H.263, VP6 and H.264.

Table 4.1 Comparison of different flash video standards [34] [38]

Feature	H.263 Baseline	VP6	H.264 Baseline
Picture type	I, P	I, P	I, P
Transform Size	8x8	8x8	4x4
Transform	DCT	Integer DCT	Integer DCT
Intra Prediction	None	Only DC mode	Yes
Motion Compensation Block Size	16x16, 8x8	16x16, 8x8	16x16, 16x8, 8x16, 8x8, 8x4, 4x8, 4x4
Total MB Modes	4	10	7 inter + (9 + 4) intra
Motion Vector resolution	½ pixel	¼ pixel	¼ pixel
Deblocking filter	None	Yes	Yes
Reference Frames	1	Max 2	Multiple

VP6 does not make use of bidirectional prediction (section 3.5). So there are no B-frames in VP6; H.264 baseline profile does not have bidirectional prediction. Hence there is no difference in display order and coding order for VP6. Since no reordering is required, delay from reordering can be avoided.

For the P-frames VP6 does not make use of multiple frames for motion estimation / motion compensation as compared to H.264. Due to the presence of multiple prediction frames in H.264 the prediction frame buffers need to be larger. Also presence of more than one prediction frame allows weighted prediction. VP6 does not support weighted prediction [34]. VP6 needs only two frame buffers for prediction - one for previous frame prediction and another for golden frames. There is no golden frame in H.264. The absence of bidirectional prediction

and weighted prediction in VP6 reduces the complexity up to 1/2 compared to the same in H.264 [34].

H.264 has up to 9 intra-prediction modes (section 2.3.1). The intra-prediction modes in H.264 make use of adjacent pixel redundancy. Intra-prediction contributes to significant increase in complexity of H.264. Intra prediction in VP6 does not have multiple modes and as a result it is much simpler compared to H.264; it only has a low cost DC prediction as noted earlier in section 3.6.1. However VP6 cannot achieve the significant redundancy reduction that H.264 achieves for intra-prediction. The intra-prediction modes in H.264 hike the complexity by roughly 2-16 times [34].

Inter-prediction in H.264 can be carried out for multiple MB and sub-MB sizes including 4x4, 4x8, 8x4, 8x8, 8x16, 16x8 and 16x16 (section 2.3.2). This helps H.264 achieve much finer prediction thereby reducing the bitrate. VP6 supports 16x16 and 8x8 MB sizes for motion estimation. A major amount of complexity in an encoder comes from motion estimation and compensation. So lowering the complexity in the motion estimation process in VP6 reduces the CPU utilization to a very large extent.

Both the codecs support up to 1/4 pixel accuracy for motion vectors. In order to generate sub-pixel values for sub-pixel motion search, interpolation filters are required. Whereas H.264 used 6-tap filtering for this VP6 makes use of only 2 or 4-tap filtering process. This gives it close to half complexity reduction [34].

H.264 has an in-loop deblocking filter (section 2.3.4). There is no specific filter for deblocking in VP6. However as mentioned in sections 3.7.2 and 3.8 the prediction loop filtering process helps to reduce the blocking artifacts in VP6. Whereas up to 5 tap loop filter is applied to every 4x4 block boundary of a transformed block in H.264 adaptively, the thresholded 2-4 tap prediction loop filter in VP6 is applied only on 8x8 motion blocks boundary [34]. So a significant complexity reduction can be achieved. According to On2 Technologies, the loop filtering process in H.264 is 2-4 times as complex as VP6 [34].

The DCT used in H.264 is an integer DCT. This DCT is described in section 2.3.3. VP6 uses a different integer DCT which is adaptive and aims to club as many 0 coefficients together as possible. To achieve this VP6 makes use of scan ordering. These processes are described in section 3.9. Whereas for most of the profiles, H.264 has only 4x4 integer DCT, VP6 has 8x8 integer DCT.

In VP6 BoolCoder context probabilities are adjusted at frame level compared to context probabilities being adjusted after each decoded symbol for CABAC used in H.264. The complexity for H.264 is roughly 1.25 to 1.5 times more than that in VP6 BoolCoder [34].

A lot of the tools in H.264 are applied only for certain profiles (figure 2.1). So the complexity varies a lot for all these H.264 profiles. However the main aim of adopting H.264 for Flash by Adobe is to support high quality video at higher resolutions. A significant difference in complexity can be seen for both the codecs in the encoding and decoding processes for the High Definition (HD) profiles [36].

4.3 Performance comparison

The authors in [36] present a comparison of performance for both the codecs. The comparison is done mainly from the aspect of High Definition (HD) content encoding and playback on systems of different configurations. The testing was performed with following viewpoints

- Codec performance/degradation at key data rates
- Output image quality for key framing and motion elements
- Encoding processing time, including power consumption during the process
- Playback smoothness on a variety of computer configurations

A data rate of 1.5 Mbps i.e. 1536 kbps was chosen for encoding. The audio track chosen was stereo 64 kbps at 22.05 kHz. This left 1472 kbps of data rate for video of 1280 x 720 resolution. The frame rate used was the native frame rate of the clips. Varied key frame-rates are used. VP6 encoding is done with 60 frames and 300 frames between key frames [36].

To be a good test of visual quality of the codecs at the output the clips were chosen such that they had the following two properties.

- Each clip contained a significant amount of motion - primarily pan and zoom. To be able to compare the sharpness and motion comparisons the camera paused during a portion of the clip [36].
- The clips contain both natural and unnatural elements. For instance, several clips show a bagpiper playing on an autumn hillside, where brilliant leaves cover the trees and the ground and the bagpipe forms unnaturally straight lines and angles. Another set of clips contain lumberjacks working to remove a tree that threatens a house, with bark and branches as the natural elements and a saw and flying chips as the unnatural elements [36].

The VP6-S codec with Simple profile was chosen. The VP6-S is specifically designed for high resolution playback for low-end processors. Both VP6-S and H.264 are encoded at 1500 kbps for high resolution content as mentioned earlier. To be wholesome in covering all processor and configuration capabilities 4 different machines were chosen. These were machines without accelerated graphics cards so as to ensure a good test of playback smoothness with respect to processor capabilities. The following windows and Macintosh machines were used for comparison [36].

- MacPro 8-core
- Dell Dimension 5150 with a 3.4 GHz Pentium 4
- Macbook (Blackbook) Core2Duo Dual Core
- Dell Inspiron with a 1.7 GHz Pentium 4

The results and conclusions of the tests are described in the sections 3.3.1 thru 3.3.4.

4.3.1 Encoding Time

The encoding tools used were specific to Flash encoding for all the codecs. Due to the high complexity and higher compression H.264 encoder was expected to take longer compared to encoding of VP6. However H.264 encoding could happen lot faster in comparison. The encoding times mentioned are the time to transcode a blu-ray M2T file to a Flash H.264 or a Flash VP6 file.

The encoding times for MacPro 8-core in minutes for the 720p content - 46 clips / 46 seconds average length, 1280x720@1500 kbps were as follows [36]

VP6-E: 29:39 minutes

VP6-S: 27:12 minutes

H.264: 21:07 minutes

4.3.2 Perceived Image quality

Perceived image quality for the 720p images was satisfactory for all the codecs. The codecs did not exhibit significant amount of perceived quality difference due to the data rate being sufficiently high. However with the presence of some unnatural elements H.264 can clearly edge out the VP6 codecs. The comparison of output image quality is given in the figures 4.1 thru 4.6 [36] [39].



Fig 4.1 An early frame from H.264 [36][39]

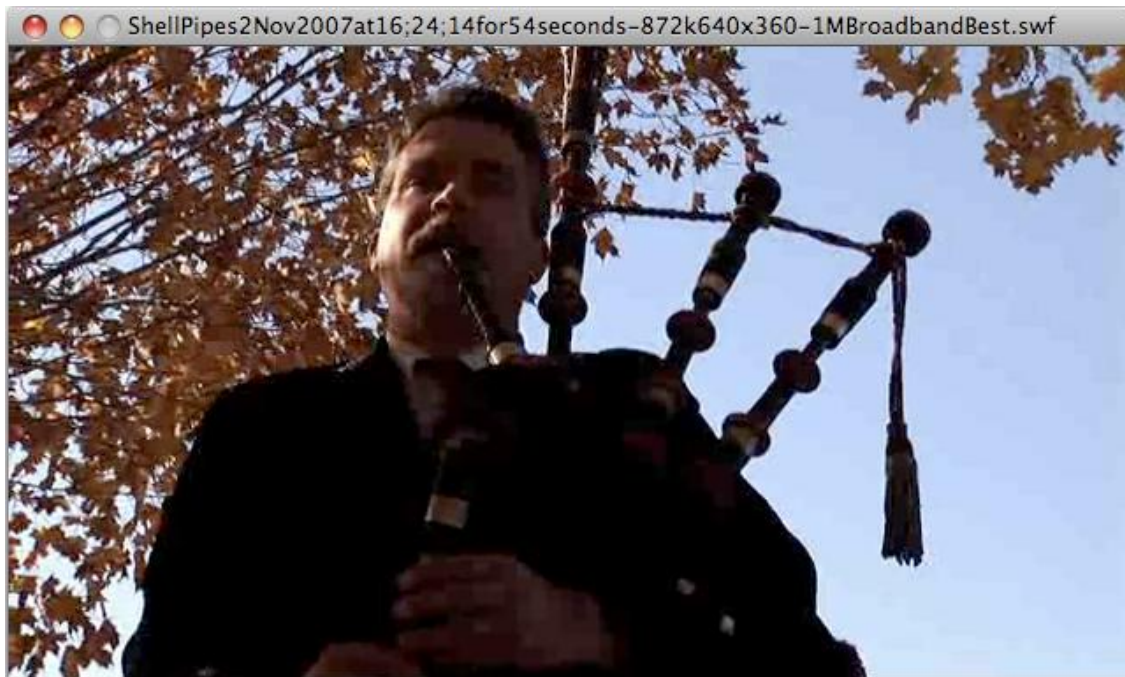


Fig 4.2 VP6 encoded with CBR [36][39]



Fig 4.3 VP6 encoded with VBR [36][39]

Figures 4.1 thru 4.3 are of an early frame in the clip. Both the natural elements such as leaves and unnatural elements such as the bagpipes are introduced in this frame. It can be observed in figure 4.1 of the H.264 clip that the quality is much higher. Although the perceived image quality difference is not very high, a limited amount of blockiness can be seen in the VP6 clips, especially in the background of the leaves. Looking closely at the bagpipe boundaries, it can be observed that the jagged angles on the bagpipe are noticeable for H.264 in comparison with the VP6 images [36].

This is an early frame in the clip, but as the video progresses the frames become more problematic as more natural and unnatural elements are introduced. Figures 4.4 thru 4.6 show the final frame from all the three codecs.



Figure 4.4 H.264 shows better quality even as VP6 starts breaking significantly [36][39]



Figure 4.5 VP6 CBR encoded frame show significant blockiness [36][39]



Figure 4.6 VP6 VBR frame is better than CBR frame but fails to match H.264 [36][39]

Figures 4.4 thru 4.6 are from the final frame in the clip. The error builds up as the video progresses. The quality difference in both the codecs is significant and very much noticeable. While VP6 fails to keep up even in the out of focus leaves background by getting blocky, the difference increases for the face especially in the low contrast regions. The VBR encoded file however has better quality although it completely fails to match up to H.264 image quality. Even more noteworthy is the difference in the unnatural object – bagpipe. The jaggy lines on pipes do not even seem to exist in VP6 images [36].

The image quality of VP6 can clearly not match the H.264. It is very easy to distinguish without using any metrics. However VP6 does exhibit advantages on other aspects.

4.3.3 Playback smoothness

A significantly notable difference can be observed in terms of the playback smoothness among the codecs. Both the VP6 codecs could very well outdo H.264 especially on the low configuration machines. Both VP6-S and VP6-E did very well on CPU utilization, although VP6-S could produce better quality images. However it was observed that in the lower configuration

machines both VP6-E and H.264 had increasing difficulties in playback. VP6-S fared out as the best playback option on the lowest of the configurations for HD playback [36].

The table 4.2 compares the playback performance by CPU utilization for different codecs on different machines.

Table 4.2 CPU usage comparison (in percentage) for different codecs (MacPro 8 cores) [36]

CPU Usage	Average	Low	High
VP6-E 448 320x180	14.3	13.4	16.9
VP6-E 872 640x360	27.8	24.8	31.2
H.264 1500 1280x720	94.0	73.0	111.1
VP6-E 1500 1280x720	68.8	60.1	72.7
VP6-S 1500 1280x720	62.1	59.8	70.2

All transcoded files played back smoothly on the Mac Pro, as did all standard-definition files on the MacBook and the Dell Dimension. HD file playback on the MacBook required reducing the frame to half size, due to the MacBook’s small screen size, to properly view the content. The Dell Dimension 5150 played the VP6-E high definition files and most of the H.264 files, but the lower-end Dell bogged down for any high-definition file other than VP6-S [36].

4.3.4 Energy efficiency in playback

As mentioned above, playing back H.264 files at 1280x720 is a very processor-intensive task. The tests were performed on a multi-core system to be sure to have enough overhead to smoothly play back the H.264 files; even the 3 GHz processor required parts of 2 cores to play back the H.264 720p files smoothly. Removing the 7.2% CPU utilization load from each test, to eliminate the idle state CPU usages for the players – a customized FLVPlayer for all Flash files and QuickTime for the H.264 files – the average CPU usage is still almost 33% higher for H.264 at equivalent data rates and resolutions than it is for VP6. The VP6-S files could hold a tighter CPU usage range, with low to high CPU utilization within roughly 8% versus H.264’s 17% range and VP6-E’s 14% range. Even on the Dell Dimension 5150 tests, the only machine other than the Mac Pro that could handle most of the H.264 files, it is found that the VP6-S files maintained this tighter range [36].

H.264 image quality is very good compared to any other codec. With good encoding tools and a powerful processor available for playback, H.264 has to be the best codec for quality in flash video. Although the choice for the codec becomes very much specific as both codecs hold their own in different areas, H.264 image quality, especially in the presence of both natural and unnatural elements is difficult to beat [36]. However considering that the playback smoothness plays a very important part in a watchable video, VP6 is here to stay along with H.264 in the Flash ecosystem [36].

4.4 Quality comparison

As seen in section 3.3 H.264 is a complex codec which takes significant amount of computations for the encoding and decoding process compared to VP6. In this section a comparison of the quality of H.264 and VP6 an encoded video sequence is done. The comparison is performed for VP6 Simple profile and H.264 Baseline profile.

The results presented here use 3 sequences (2 QCIF and 1 CIF video sequence). The source of the Foreman (QCIF) sequence is the Joint Model (JM) software package [63] and the source of the Akiyo (QCIF) and Stefan (CIF) sequences is [64]. The sequences are encoded at different bitrates and the metrics used for comparison are mean squared error (MSE), peak signal to noise ratio (PSNR) and structural similarity index metric (SSIM) [SSIM source]. Tables 4.3 thru 4.5 give the Y, U and V component MSE, PSNR and SSIM values for the Foreman, Akiyo and Stefan sequences respectively. Figures 4.7 thru 4.18 show the decoded images for each codec against the original frame with the corresponding PSNR and SSIM values for that frame and graphs for comparison of quality for VP6 and H.264 in terms of average MSE, PSNR and SSIM values vs. bitrate.

The JM software is used for the implementation of H.264 [63]. JM software is free software which provides flexibility of free development and testing to the user. It is configurable software so the encoder properties can be controlled by the user. The bitrate was varied by changing the quantization parameter (QP) for the I-frames and P-frames; no B-frames were

used as they are not allowed in the H.264 baseline profile. The frame rate used was in accordance that prescribed for the levels used (level 1.0 for QCIF video sequences and level 1.2 for the CIF video sequences). The number of reference frames used was restricted to 2 reference frames.

The VP6 software used was VP6 software development kit (SDK) available from the license owner On2 technologies. It is not a free software license. However the license was provided with no cost for study and research purposes by On2 technologies to the Department of Electrical Engineering, University of Texas at Arlington (UTA); with the condition that it is not used for commercial purposes. The VP6 bitrate can be controlled by the quality parameters *BestAllowedQ* and *Quality*.

Table 4.3 Quality metrics for Foreman clip for VP6 and H.264 codecs (PSNR in dB)

Clip – Foreman (3 frames only) [63]									
Codec – VP6									
Bitrate (kbps)	Y MSE	U MSE	V MSE	Y PSNR	U PSNR	V PSNR	Y SSIM	U SSIM	V SSIM
10.723	52.734	8.267	7.833	31.015	38.988	39.201	0.877	0.941	0.955
14.199	33.898	7.067	5.808	32.923	39.667	40.515	0.908	0.945	0.964
17.251	26.658	6.380	4.948	33.992	40.110	41.209	0.922	0.946	0.966
20.894	19.695	5.527	4.166	35.284	40.720	41.953	0.935	0.952	0.969
30.518	12.131	3.928	2.789	37.470	42.221	43.722	0.954	0.961	0.976
45.249	6.980	2.643	1.747	39.832	43.960	45.773	0.969	0.972	0.983
64.629	4.250	1.845	1.286	41.852	45.473	47.039	0.978	0.979	0.987
Codec – H.264 baseline									
Bitrate (kbps)	Y MSE	U MSE	V MSE	Y PSNR	U PSNR	V PSNR	Y SSIM	U SSIM	V SSIM
12.930	17.044	5.599	3.620	31.418	39.276	40.622	0.891	0.944	0.960
17.261	17.008	5.569	3.611	35.210	40.199	41.409	0.940	0.950	0.964
45.098	5.180	2.918	1.894	41.151	43.525	45.417	0.977	0.969	0.982
109.966	2.783	1.565	1.121	47.049	47.865	48.869	0.988	0.983	0.989
199.678	1.082	0.716	0.457	57.599	55.522	55.816	0.995	0.992	0.995

As it can be observed for the range of PSNR and SSIM values in table 4.3 the quality of H.264 increases significantly compared to VP6 at higher bitrates. At lower bitrates the difference in values is not so significant. Figures 4.7 gives a comparison of the I-frame image from the VP6 and H.264 files for the Foreman clip.

VP6 (Bitrate - 230.352 kbps)

Y comp PSNR - 53.849 Foreman - Original Sequence
 Y comp SSIM - 0.9983 (1st frame - I frame)

H.264 (Bitrate - 199.678 kbps)

Y comp PSNR - 67.486
 Y comp SSIM - 0.9999



(a)

(b)

(c)

Figure 4.7 Foreman clip (1st frame) (a) VP6 (b) original (c) H.264 (PSNR in dB)

Table 4.4 Quality metrics for Akiyo (QCIF) clip for VP6 and H.264 (PSNR in dB)

Clip – Akiyo (30 frames at 15 fps) [54]									
Codec – VP6									
Bitrate (kbps)	Y MSE	U MSE	V MSE	Y PSNR	U PSNR	V PSNR	Y SSIM	U SSIM	V SSIM
18.768	34.667	15.393	9.021	32.757	36.262	38.579	0.900	0.949	0.942
26.544	22.465	9.463	6.342	34.647	38.388	40.111	0.926	0.960	0.952
33.968	16.851	6.631	4.903	35.909	39.927	41.235	0.942	0.967	0.960
36.280	14.132	6.182	4.336	36.698	40.243	41.769	0.949	0.968	0.965
61.856	7.821	2.917	2.419	39.263	43.489	44.306	0.967	0.981	0.977
92.072	5.026	2.000	1.572	41.194	45.145	46.186	0.975	0.986	0.984
198.984	2.840	1.247	1.080	43.603	47.176	47.802	0.982	0.990	0.989
378.880	1.896	0.725	0.653	45.358	49.527	49.986	0.986	0.993	0.993
682.536	1.466	0.461	0.423	46.476	51.493	51.873	0.988	0.995	0.995
Codec – H.264 baseline									
Bitrate (kbps)	Y MSE	U MSE	V MSE	Y PSNR	U PSNR	V PSNR	Y SSIM	U SSIM	V SSIM
719.820	0.035	0.054	0.055	64.479	61.715	61.488	1.000	0.999	0.999
479.790	0.238	0.256	0.244	55.461	54.657	54.690	0.998	0.997	0.997
267.100	0.554	0.541	0.498	50.950	50.947	51.251	0.996	0.995	0.995
58.620	3.140	2.218	1.798	43.193	44.685	45.594	0.985	0.984	0.983
15.700	12.876	5.837	4.923	37.040	40.469	41.211	0.961	0.970	0.960
10.820	19.242	6.056	5.004	35.459	40.315	41.139	0.953	0.970	0.960
10.100	26.396	6.330	5.089	34.404	40.138	41.068	0.946	0.970	0.960

Table 4.4 gives the values for different metrics for the Akiyo clip. The initial 30 frames of the clip were encoded at 15 frames per second. It can be observed the quality of the H.264 encoded frames sequence is better than VP6 encoded sequence. Figures 4.8 and 4.9 compare the I-frame (1st frame) and the 30th frame which was the last frame in the sequence. It can be observed that the error is high for 30th frame.

VP6 (Bitrate - 10.648 kbps)	Akiyo - Original sequence	H.264 (Bitrate - 10.82 kbps)
Y comp PSNR - 35.577	(1st frame - I frame)	Y comp PSNR - 37.434
Y comp SSIM - 0.9370		Y comp SSIM - 0.9621



(a) (b) (c)

Figure 4.8 Akiyo clip (1st frame) (a) VP6 (b) original (c) H.264 (PSNR in dB)

VP6 (Bitrate - 10.648 kbps)	Akiyo - Original sequence	H.264 (Bitrate - 10.82 kbps)
Y comp PSNR - 33.565	(30th frame)	Y comp PSNR - 33.799
Y comp SSIM - 0.9155		Y comp SSIM - 0.9437



(a) (b) (c)

Figure 4.9 Akiyo clip (30th frame) (a) VP6 (b) original (c) H.264 (PSNR in dB)

The graphs in figures 4.10 thru 4.12 compare the average values of MSE, PSNR and SSIM for the Y-component over 30 frames of the sequence.

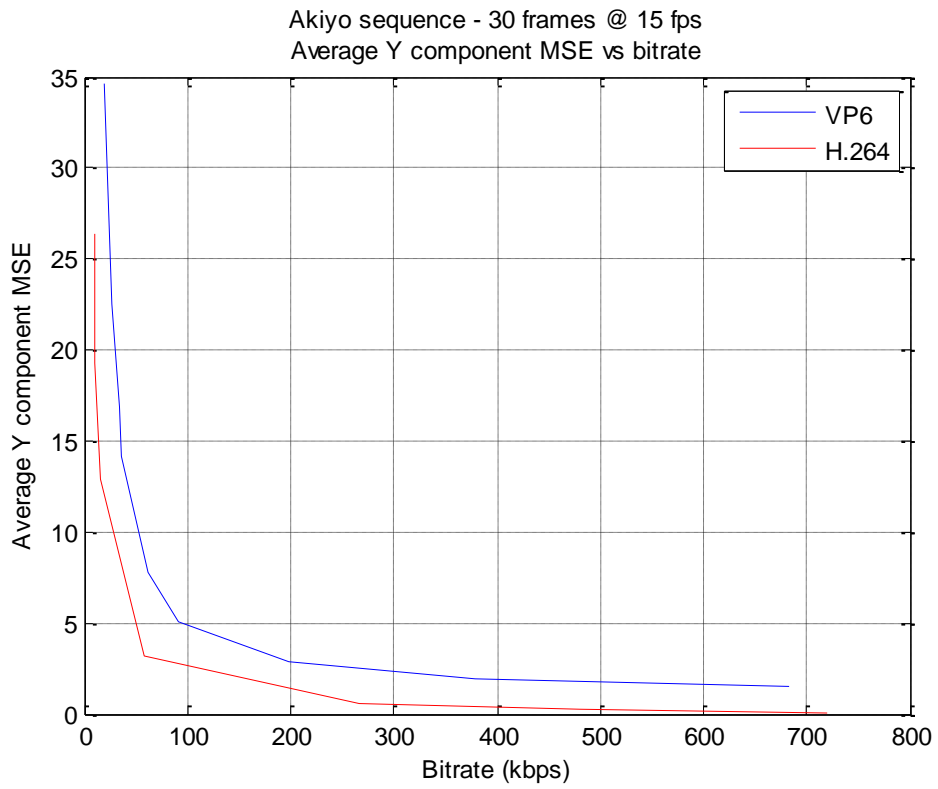


Figure 4.10 Average Y component MSE vs. bitrate for the Akiyo sequence

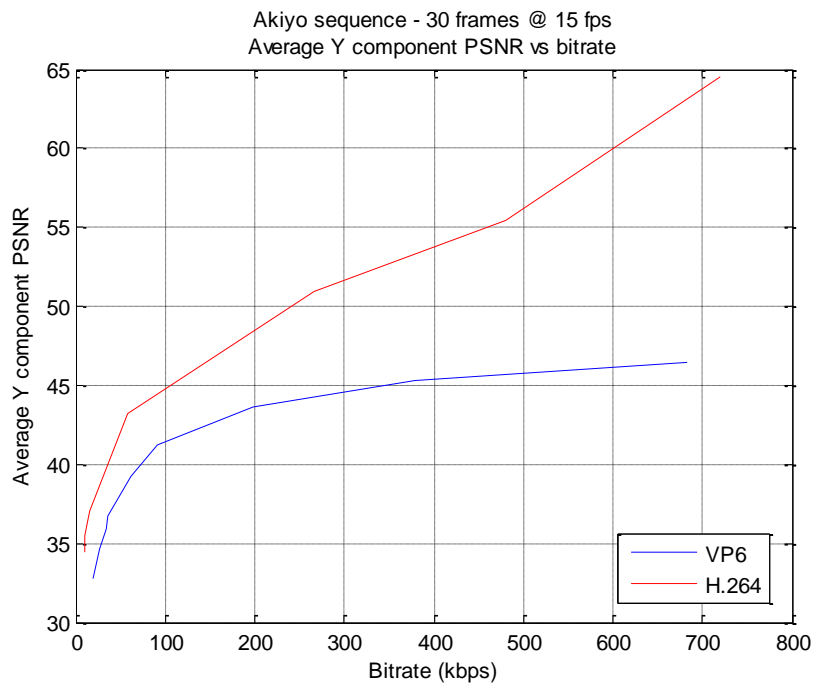


Figure 4.11 Average Y component PSNR (dB) vs. bitrate for Akiyo sequence

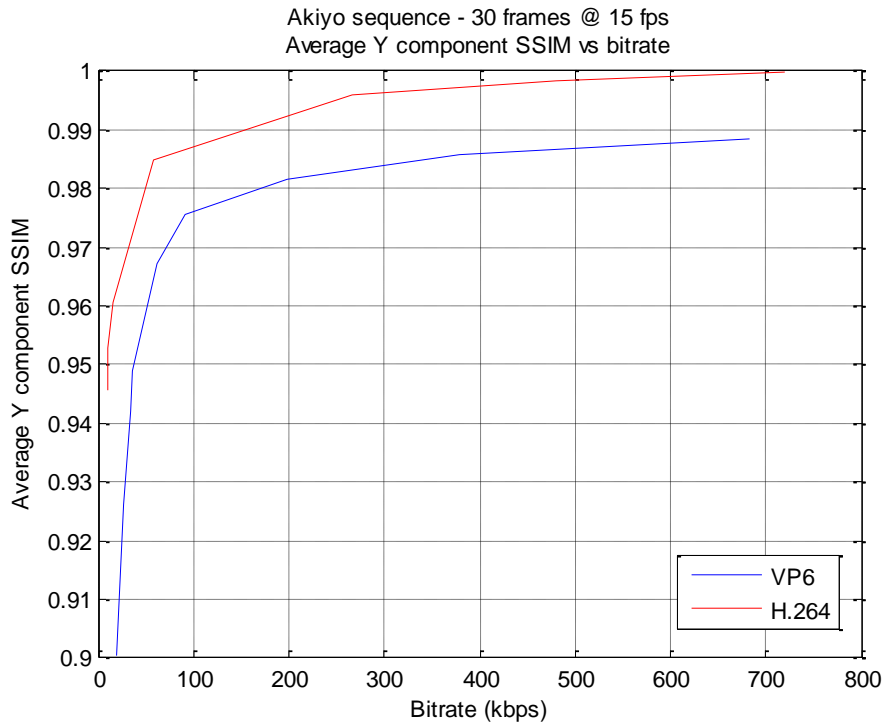


Figure 4.12 Average Y component SSIM vs. bitrate for Akiyo sequence

Table 4.5 gives the values of different metrics for the Stefan sequence. It is CIF sequence. The initial 15 frames of the sequence were encoded at 15 fps with only one I-frame at the first frame. As it can be observed there is not a significant difference in the quality for both the codecs for this sequence.

Table 4.5 Quality metrics for Stefan sequence (PSNR in dB)

Clip – Stefan (15 frames at 15 fps) [64]									
Codec – VP6									
Bitrate (kbps)	Y MSE	U MSE	V MSE	Y PSNR	U PSNR	V PSNR	Y SSIM	U SSIM	V SSIM
164.696	157.906	26.667	29.772	26.403	33.933	33.460	0.838	0.841	0.838
285.248	86.330	20.517	21.904	28.925	35.081	34.819	0.898	0.867	0.873
362.048	64.973	16.819	17.843	30.164	35.959	35.723	0.918	0.889	0.892
543.488	39.440	12.450	12.820	32.276	37.238	37.117	0.942	0.914	0.920
834.792	21.889	8.481	8.617	34.846	38.926	38.863	0.962	0.940	0.945
1445.568	9.263	4.568	4.543	38.518	41.593	41.621	0.977	0.965	0.969
5685.424	0.521	0.495	0.484	50.965	51.185	51.279	0.997	0.995	0.995

Table 4.5 – Continued

Codec – H.264 baseline									
Bitrate (kbps)	Y MSE	U MSE	V MSE	Y PSNR	U PSNR	V PSNR	Y SSIM	U SSIM	V SSIM
160.520	214.934	19.323	20.370	25.516	35.416	35.206	0.839	0.889	0.893
221.060	74.135	16.532	17.608	29.695	36.049	35.786	0.927	0.899	0.902
290.130	51.605	16.903	17.522	31.008	35.855	35.696	0.943	0.891	0.896
601.940	23.064	11.443	11.323	34.637	37.572	37.623	0.968	0.920	0.929
611.710	20.647	10.454	10.416	34.990	37.944	37.957	0.971	0.927	0.935
1242.820	10.167	6.504	6.439	38.911	40.199	40.261	0.982	0.954	0.958
2179.000	3.247	2.728	2.667	43.146	43.835	43.945	0.990	0.979	0.981
5514.090	0.871	0.764	0.761	50.937	50.620	50.698	0.997	0.993	0.994

Figure 4.13 and 4.14 provide a comparison for the 2nd frame which is the first frame after the I-frame and 15th frame for the sequence. The graphs in figures 4.15 thru 4.17 compare the values average metrics for the Y-component against bitrate in kbps.

Stefan (Original CIF clip)
Encoding: 15 frames @ 15 fps



VP6 (Bitrate - 660.9 kbps)
Y comp PSNR - 34.996
Y comp SSIM - 0.9651



H.264 (Bitrate - 611.7 kbps)
Y comp PSNR - 34.225
Y comp SSIM - 0.9674



Figure 4.13 2nd frame from the Stefan (CIF) clip (PSNR in dB)

Stefan (Original CIF clip)
Encoding: 15 frames @ 15 fps
frame 15



VP6 (Bitrate - 660.9 kbps)
Y comp PSNR - 34.23
Y comp SSIM - 0.9588



H.264 (Bitrate - 611.7 kbps)
Y comp PSNR - 35.028
Y comp SSIM - 0.9719



Figure 4.14 15th frame from the Stefan (CIF) clip (PSNR in dB)

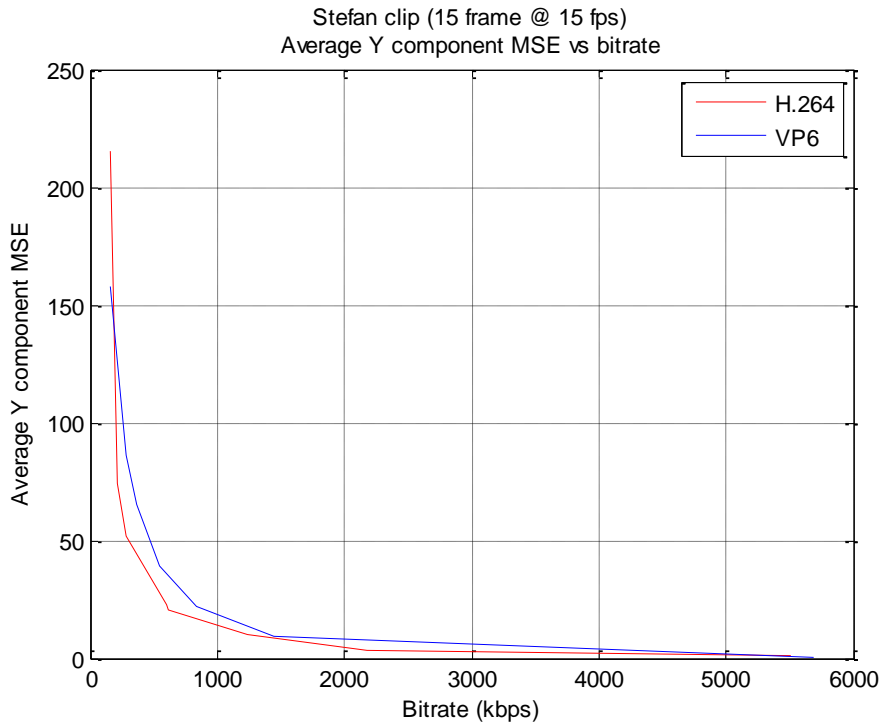


Figure 4.15 Average Y component MSE vs bitrate for the Stefan (CIF) clip

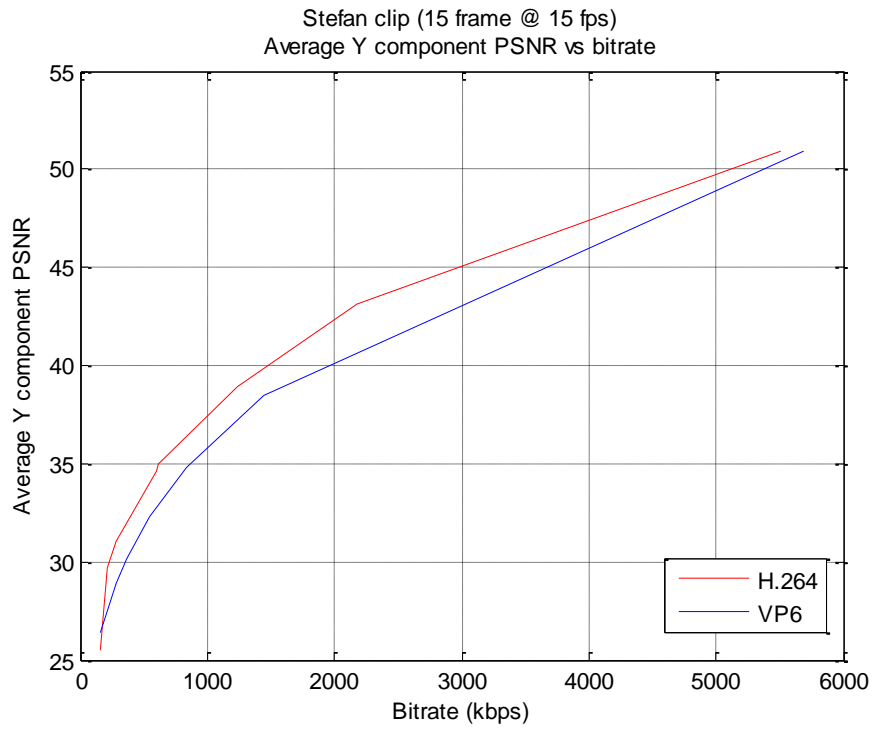


Figure 4.16 Average Y component PSNR (dB) vs bitrate for Stefan (CIF) clip

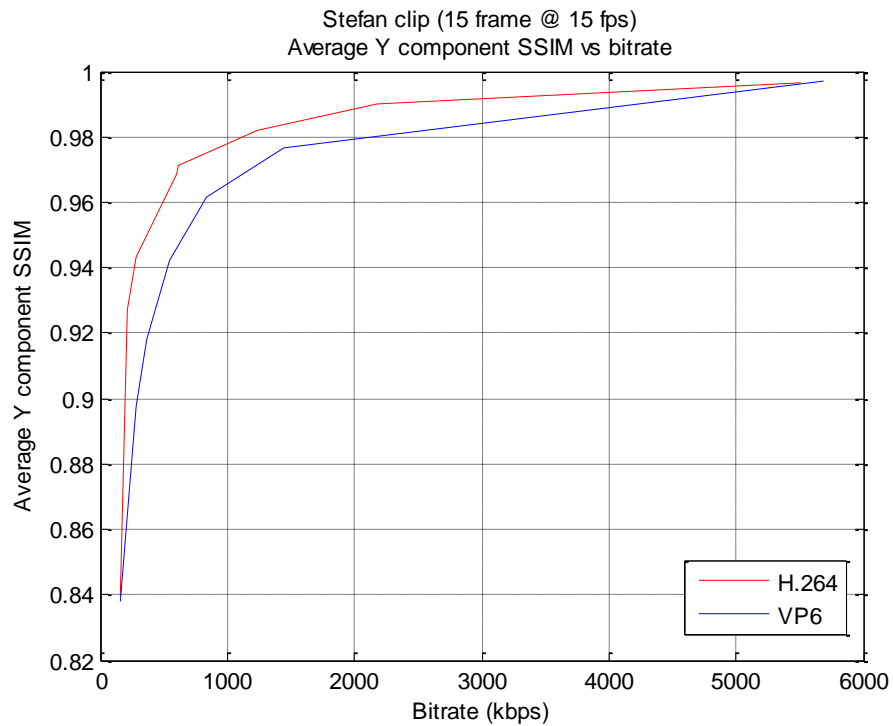


Figure 4.17 Average Y component SSIM vs bitrate for the Stefan clip

4.5 Summary

The chapter gives an overview of the difference between the encoding tools and compares the complexity for both the codecs. Further performance comparison is given with an overview subject quality assessment. The last section in the chapter provides a comparison of object quality assessment using metrics such as MSE, PSNR and SSIM against the bitrate. It can be observed that while H.264 can provide better quality at the same bitrate its complexity is higher and it is more strenuous on the processor.

CHAPTER 5
TRANSCODING
5.1 Introduction

As described in section 1.2 video transcoding is an operation of converting video from one format to another [31]. This format conversion includes a range of operations such as bitrate reduction, conversion of one compression format to another, altering video container format or changing the header descriptions and others. Apart from this basic format conversion, a transcoder can be used for other functions such as adjustment of coding parameters of compressed video, spatial and temporal resolution conversions, insertion of new information such as digital watermarks or company logos and even enhanced error resilience [30].

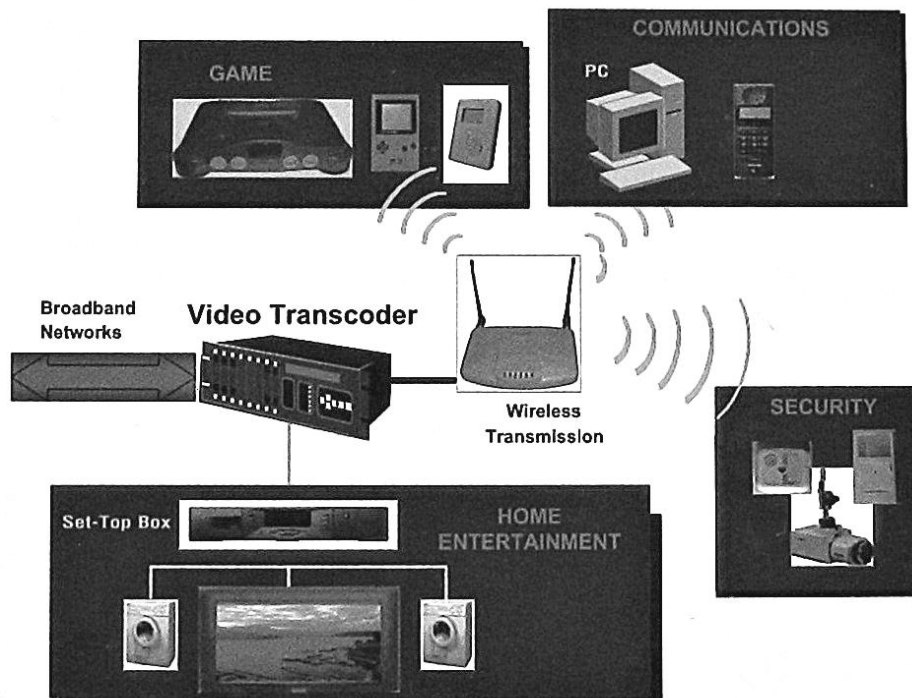


Figure 5.1 Communication between various multimedia devices [39]

Figure 5.1 shows different multimedia devices; these devices operate at different bitrates, picture resolutions and video coding algorithms. Due to the scope of communications in current times there is extensive exchange of multimedia data between various such systems of different configurations over networks of varied capacities. It is very important to be able to play the same video content for all the systems to ensure access to a widespread audience. Transcoding can enable multimedia devices of diverse capabilities and formats to exchange video content on heterogeneous network platforms [30]. Figure 5.2 is a high level look on transcoder functionality. High bitrate video from high quality sources such as DVD and HDTV broadcasting can be made available on hand held devices such as smartphones, PDAs, etc. In video conferencing transcoding can enable the adjustment of bitrate to support the bandwidth requirements and also any format conversions if required. This attempt to ensure availability to all media content over diverse networks and systems with varied capabilities forms an important aspect of universal multimedia access [30] [31].

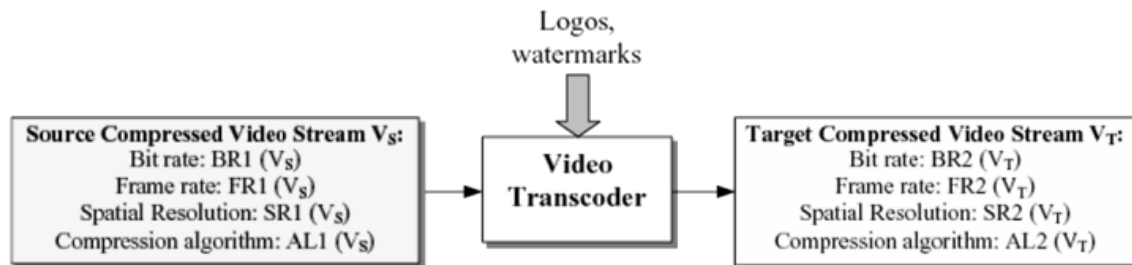


Figure 5.2 Video transcoding operations [29]

Several video coding standards exist currently [5] [23] [27] [33]. Each of them suits a diverse range of applications, but is optimized for certain kind of applications. This results in the use of various standards over diverse applications. Table 5.1 lists the different codecs used for different multimedia applications. Some of the standards are proprietary and the IP owners prefer using their standards. Some of the common standards can be listed as - H.261, H.262 H.263, H.263+ [33] designed by ITU (International Telecommunication Union); they are aimed at low-bit-rate video applications such as videophone and videoconferencing. MPEG standards

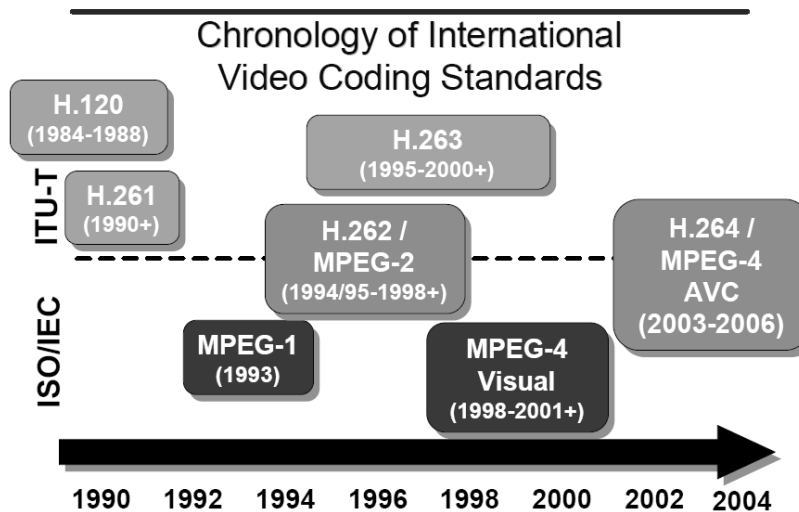
are defined by ISO (International Organization for Standardization). MPEG-2 is aimed for high bit rate high quality applications such as digital TV broadcasting and DVD, and MPEG-4 is aimed at multimedia applications including streaming video applications on mobile devices [29]. The new and highly popular H.264 standard is used for high resolution video content and a range of broadcast applications. This standard addresses a very wide variety of applications [5] [40]. Apart from that, Microsoft developed the Windows Media Video standard WMV9 which is adopted by SMPTE as VC-1 standard and is popular [23]. VC-1 along with H.264 and MPEG-2 is also used for high definition content in Blu-ray Disc standard [41]. Adobe licenses the VP6 and VP7 codecs from On2 Technologies, Inc. for its flash tools [25]. Thus a wide range of standards exist. Figure 5.3 provides an overview of how these standards have grown over the years and the increase in complexity along with that. Transcoding one video format to another is increasingly significant in the current scenario of diversifying and increasing multimedia applications. The current research on VP6 and H.264 standard is aimed at the interoperability of the two popular standards supported by Adobe in its Adobe Flash Player.

Table 5.1 Different multimedia applications and corresponding video standards [42][41]

Application	Bitrate	Video standard
Digital TV broadcasting	2 to 6 Mbps (10 to 20 Mbps for HD broadcast)	MPEG-2
DVD Video	6 to 8 Mbps	MPEG-2
Internet video streaming	20 to 200 kbps	Flash – Sorenson spark (based on H.263), VP6 and H.264; Silverlight uses VC-1; and also MPEG-4 Part 2
Video conferencing and video-telephony	20 to 320 kbps	H.261, H.263, H.263+
Video over 3G wireless	20 to 100 kbps	H.263, MPEG-4. Part 2
High definition – Blu-ray and HD-DVD	36 to 54 Mbps	H.264, VC-1 and MPEG-2

Apart from this, transcoding is useful in a range of other applications. In statistical multiplexing [44], the bit rate increases as various multi-bit-rate video streams are multiplexed. A transcoder can be used to adapt the bit-rates of the video streams when the aggregated bit-rate exceeds the channel bandwidth [30]. A transcoder can also be used to insert new

information including company logos, watermarks, as well as error-resilience features into a compressed video stream. Transcoding techniques are also useful for supporting VCR trick modes, i.e., fast forward, reverse play, etc., for on-demand video applications [30]. For adaptive video content delivery, object based transcoding techniques can be used [45][30].



(a)

- *Intraframe coding*: only spatial correlation exploited
→ DCT [Ahmed, Natarajan, Rao 1974], JPEG [1992] Complexity increases
- *Conditional replenishment, DPCM, scalar quantization*
→ H.120 [1984]
- *Frame difference coding*
→ H.120 Version 2 [1988]
- *Motion compensation: integer-pel accurate displacements*
→ H.261 [1991]
- *Half-pel accurate motion compensation*
→ MPEG-1 [1993], MPEG-2/H.262 [1994]
- *Variable block-size motion compensation*
→ H.263 [1996], MPEG-4 [1999]

(b)

Figure 5.3 Video coding standards (a) Timeline [43] (b) Increase in complexity [42]

Transcoding can be broadly categorized into homogenous and heterogeneous transcoding techniques based on applications. Figure 5.4 shows the classification of different transcoding techniques.

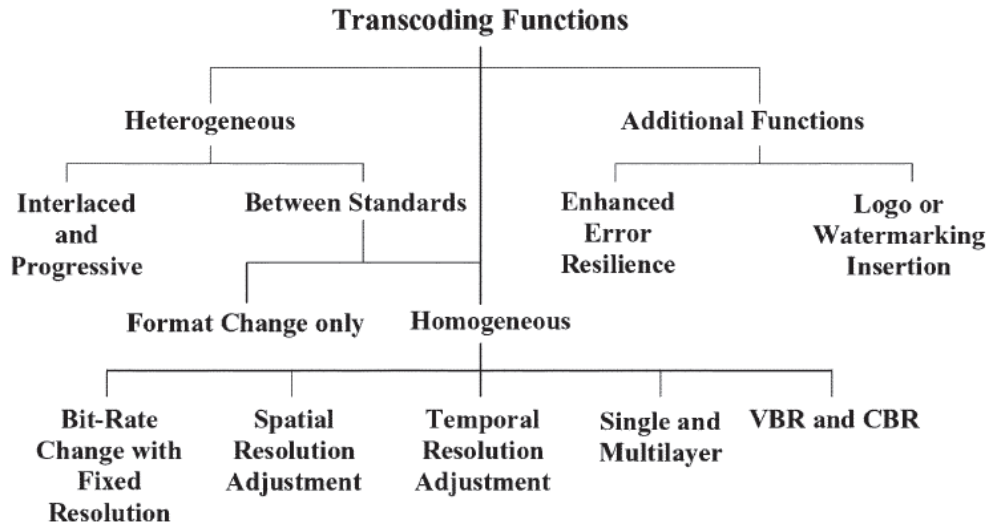


Figure 5.4 Video transcoding operations and classification [29]

Homogenous transcoding performs conversion between video bitstreams of the same standard. Bitrate conversion is one such application. A simple technique to achieve bitrate reduction can be to increase the quantization step at the encoder part of the transcoder [29] [46] [47] (section 5.2.2.3). As the quantization resolution decreases, the number of non-zero coefficients decreases, thereby resulting in bitrate reduction. The complexity of such an application is less, although the reconstructed image quality can be affected; it is considered as a good trade-off. Another method of fixed resolution bit rate reduction can be selective transmission (section 5.2.2.2). Since most of the energy is concentrated in the lower frequency bands of an image, discarding (truncating) some of the higher frequency coefficients [39] [29] can be used. This can preserve picture quality, but can introduce blocking artifacts in the reconstructed target video. Various other methods for bitrate transcoding at architectural levels are discussed in sections 5.2 and further.

Other application of homogenous transcoding is spatial or temporal transcoding. As shown in figure 5.5 [29], spatial transcoding can be implemented in various ways which can both achieve spatial as well as bit rate adjustments. There can be multiple uses of performing spatial transcoding such as to subsample or even to extract sections of the image to user's interest as shown in figure 5.6 [29]. This requires the use of meta information. In subsampling, filtering and pixel averaging to reduce spatial resolution [29] problems arise when passing motion vectors directly from the decoder to the encoder. Thus, motion vectors need to be refined [32] [37] (section 5.2.5.4). In [48] the authors proposed a filter that can be used in both horizontal and vertical directions for luminance and chrominance; the image is then down-sampled by dropping every alternate pixel in both horizontal and vertical directions (section 5.2.5.1). In pixel-averaging (section 5.2.5.2) [48], MxM pixels are represented by averaging their values to a single pixel. It is a very simple method, but the the reconstructed pictures may become blurred. In [49] spatial resolution reduction is achieved by performing decimation in DCT domain by discarding the higher order DCT coefficients (section 5.2.5.3).






	WorkStation/LAN	PC/Dialup	TV Browser	Gray PDA	B/W PDA
Video					
Color	24 bit color	24 bit color	256 colors	4 bit gray	B/W
Size	352×288	176×144	176×144	160×120	128×96
Bits	64KB	33KB	8KB	4KB	0.6KB

Figure 5.5 Various ways of spatial transcoding (Bits = target bitrate in KBps) [29]

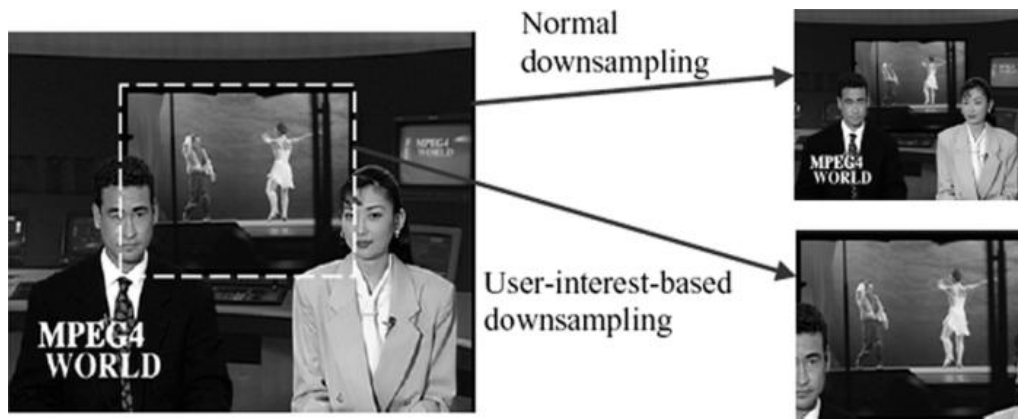


Figure 5.6 Transcoding with normal downsampling and user-interest-based downsampling [29]

Frame-rate conversion is needed when the end-system supports only a lower frame-rate. Reduction in frame rate may save bits that can be used in the remaining frames to maintain acceptable overall picture quality for each frame. However this requires motion vector recalculation; with the dropped frames, incoming MVs may not be valid as they point to frames that do not exist. In [50] a method to estimate the new MVs by using bilinear interpolation is described. Given the MVs between every adjacent dropped frames are known, bilinear interpolation (section 5.2.6.1) is used to calculate the new MVs between the current and previous non-skipped frame. Another method proposed in [51] known as the Forward Dominant Vector Selection (FDVS) (section 5.2.6.2) selects dominant MV from the four neighboring macroblocks. A dominant MV is defined as the MV carried by a macroblock that has the largest overlapping segment with the block pointed by the incoming MV. The Telescopic Vector Composition (TVC) technique (section 5.2.6.3) described in [48] sums up all the MVs of the corresponding macroblocks of the dropped frames and adds the resulting combined MV to its corresponding MV in the current frame. This technique also carries out new macroblock decision and MV refinement. Another algorithm known as Activity-Dominant Vector Selection (ADVS) (section 6.2.6.4) is described in [52]. It utilizes the activity of the macroblock to decide the choice of the MV. The activity information of a macroblock is represented by counting the

number of nonzero quantized DCT coefficients of covered 8x8 residual blocks, other statistics, such as the sum of the absolute values of DCT coefficients, etc.

A heterogeneous video transcoder provides conversions between two different video coding standards. This involves basic syntax conversion as the first step. Further it can also provide all the other functionalities of a homogenous transcoder.

5.2 Video transcoding architectures

5.2.1 Cascaded decoder and encoder model

The most straightforward transcoding architecture is to cascade the decoder and encoder directly as shown in Fig. 5.7 (a) [29]. In this architecture, the incoming source video stream (V_S) after encoding is fully decoded, and then the decoded video is re-encoded into the target video stream (V_T) with desirable bit-rate or format; the process of transcoding does not introduce any degradation in the visual quality. The more detailed manifestation of the cascaded transcoder is shown in figure 5.8 (b) [29].

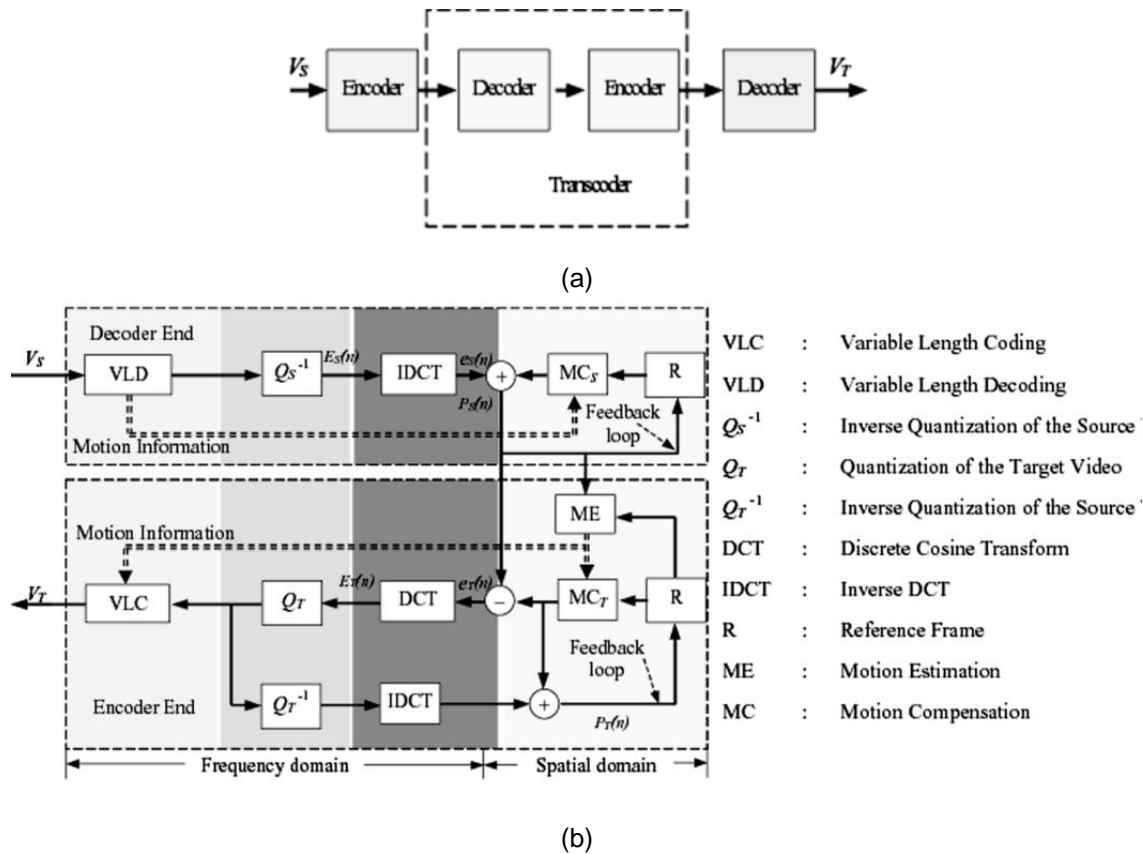


Figure 5.7 Cascaded decoder and encoder model (a) block level diagram
(b) detailed diagram [29]

This type of implementation involves complete decoding and re-encoding of the incoming compressed video stream. It has to perform full decoding followed by the resizing / re-ordering of the decoded sequence before re-encoding it. Due to complete re-encoding operation, complex frame reordering and full-scale motion re-estimation are required. Motion estimation has the highest complexity in the encoder. So such an implementation involves the maximum complexity and also high processing time and power consumption leading to significant delays [53] [39]. Also the pictures / frames exhibit increased error due to re-encoding being performed on decoded pictures which have lower quality than original frames. The error is due to propagation. Lossy encoding process inserts errors; when such a bitstream is decoded, the decoded pictures have errors which propagate on further encoding which inserts more

errors. This error is not the same as the drift error in open loop transcoders described in section 5.2.2.3. Due to all these reasons such a transcoding model needs a lot of optimization. Different methods are described further to optimize transcoder performance and reduce complexity [53].

5.2.2 Open loop transcoding architecture

The simplest method to reduce the complexity of the cascaded decoder encoder transcoding model is to use open-loop transcoding architecture. Such architecture aims to use minimum transcoding complexity by only modifying the encoded DCT coefficients. Figure 5.8 shows an open loop transcoder. Since only the DCT coefficients are modified in order to adjust the bitrate, other video parameters remain unaffected; the DCT coefficients are decoded while the rest of the parameters are transmitted in VLC domain. After the DCT coefficients are decoded, operations may be performed on them to reduce the bitrate. Bitrate reduction can be achieved by either throwing away the high frequency components or coarser re-quantization of the decoded coefficients. These schemes are discussed in section 5.2.2.2 and 5.2.2.3 respectively. This architecture is called so because it does not have any feedback loop which can compensate the drift errors (section 5.2.2.1 describes drift errors and reduction of the drift errors using a feedback loop). The picture drift occurs from the mismatch between the locally reconstructed pictures at the encoder and the transcoded pictures in the system.

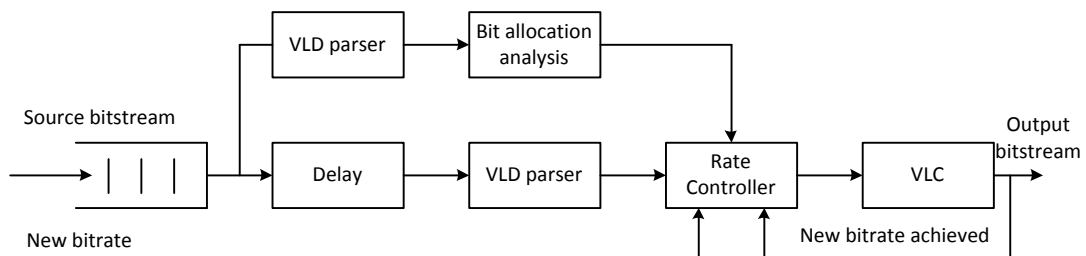


Figure 5.8 Open loop transcoder architecture [39]

5.2.2.1 Picture drift errors

A cumulative effect occurs due to the mismatch between the reconstructed images of originally encoded video frames and the transcoded video frames. Such errors propagate along the video sequence. The errors resulting from mismatch between the video frames at the original encoder and the eventual decoder are known as drift errors and the effect is known as the picture drift effect in a transcoder [29] [53]. The following sets of equations [29] develop drift errors mathematically. All the symbols in the equations below have reference in the figure 5.7 (b).

$$P_S(n) = e_S(n) + MC_S(P_S(n-1)) \quad (5.1)$$

$$e_T(n) = P_S(n) - MC_T(P_T(n-1)) \quad (5.2)$$

$$\begin{aligned} E_T(n) &= DCT(e_T(n)) \\ &= DCT(P_S(n) - MC_T(P_T(n-1))) && \text{[from equation 5.2]} \\ &= DCT(e_S(n) + MC_S(P_S(n-1)) - MC_T(P_T(n-1))) && \text{[from equation 5.1]} \\ &= DCT(e_S(n)) + DC_T(MC_S(P_S(n-1)) - MC_T(P_T(n-1))) \\ &= E_S(n) + DCT(\Delta MC) \end{aligned} \quad (5.3)$$

where

$$\Delta MC = MC_S(P_S(n-1)) - MC_T(P_T(n-1))$$

Equation 5.3 shows that if ΔMC is non-zero i.e. the video frames at the original encoder and the eventual decoder are different, the transcoder output has errors compared to the original input. These errors build up as the video frames progress. As observed the errors arise during motion compensation (MC). As there is no motion compensation in intra frames, drift errors do not occur in intra-frames. However as each predicted frame (P-frame) makes use of another P-frame which already has drift errors, the errors build up. So drift error keeps increasing with each frame until a new I-frame is reached. B-frames are not used for prediction. So they do not contribute to further increase the drift errors [29] [53].

The feedback loop in figure 5.7 (b) makes sure the building up of drift errors can be avoided. The feedback loop can ensure that for further prediction the MC block at the encoder uses the same frame as the MC block of the decoder thereby preventing the building up of drift errors due to erroneous prediction frames.

5.2.2.2 Truncation of high-frequency coefficients

Figure 5.9 shows an open-loop transcoder which reduces the bitrate by discarding the high frequency coefficients. The variable length decoder (VLD) decodes the DCT coefficients and based on the target bit-rate defines a scaled bit usage profile to meet the target rate. The rate controller simply has to discard the coefficients that exceed the scaled profile. For this most often the VLD only needs to decode the codeword lengths. This method does not need to perform inverse quantization and re-quantization as the method described in section 5.2.2.3. It is the simplest implementation of a bit-rate transcoder.

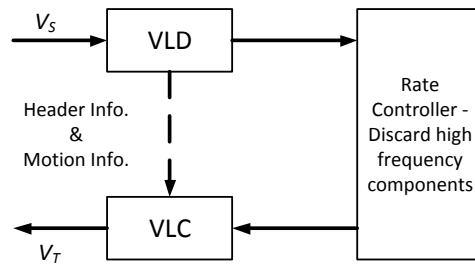


Figure 5.9 Bitrate reduction by truncation of high frequency coefficients [29]

5.2.2.3 Re-quantization to reduce bitrate

Figure 5.10 shows an open-loop transcoder which employs the method of coarser re-quantization of DCT coefficients in order to achieve reduction in bitrate. Compared to the architecture in section 5.2.2.2 it has a complete VLD block, an inverse quantizer and a quantizer; all these contribute towards increase in the complexity slightly. The DCT coefficients are decoded first, inverse quantized, re-quantized with a coarser quantization value and variable length coded (VLC) again. This architecture also exhibits drift errors as there is no

feedback loop. The equations below give an analysis of the drift error in open-loop architecture.

All the symbols used in the equations are in reference with figure 5.10

$$E'_T(n) = E_S(n)$$

$$E_T(n) = E'_T(n) + \text{DCT}(\Delta\text{MC}) \quad [\text{from equation 5.3}]$$

$$d_e(n) = \text{DCT}(\Delta\text{MC}) = E_T(n) - E'_T(n) \quad (5.4)$$

It can be seen from equation 5.4 that here $d_e(n)$ is the drift error which is introduced by the process of re-quantization. Similarly in the scheme in section 5.2.2.2 such an error is introduced by the discarding on non-zero high frequency coefficients.

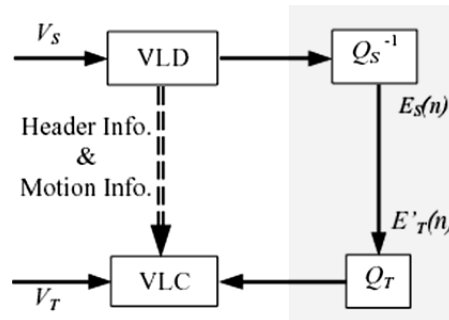


Figure 5.10 Transcoding with re-quantization scheme [29]

5.2.3 Spatial domain transcoding architecture

Figure 5.7(b) shows a spatial-domain transcoding architecture (SDTA). It is the most basic architectures to perform bit-rate reduction; but as described in section 5.2.1, it has a very high complexity. Figure 5.11 shows a slightly modified SDTA which makes reuse of the motion information from the video decoder. This results in significant reduction in complexity as motion estimation alone contributes to 60-70% of encoder time complexity [29]. As it can be observed, there are optional functional blocks between the decoder and the encoder in figure 5.11. Motion vector composition and refinement (MVCR) and spatial/temporal resolution reduction (STR) can be performed at these blocks. Different methods to achieve these are explained in section 5.2.5.4 and section 5.2.5 respectively. If the only aim of the SDTA is bit-rate reduction the transcoder can be further simplified as shown in figure 5.12 [29].

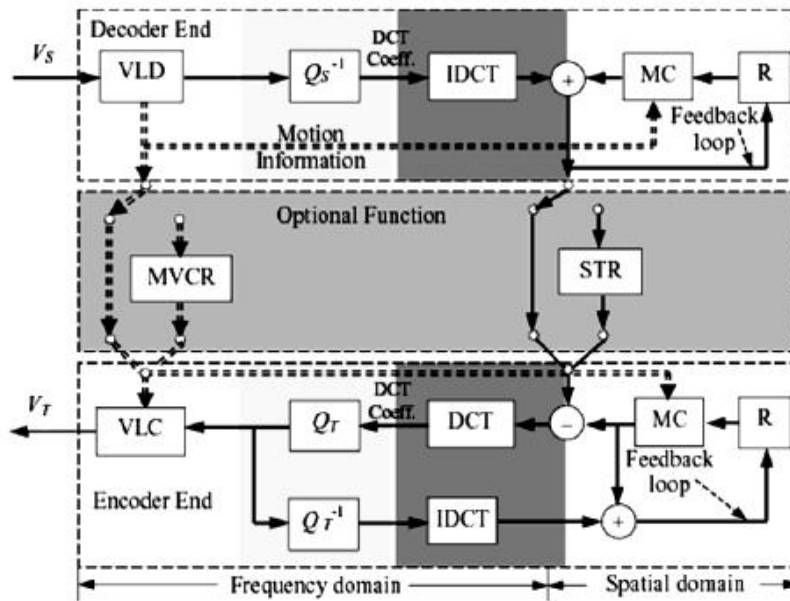


Figure 5.11 Spatial domain transcoding architecture (SDTA) with MV reuse and STR [29].

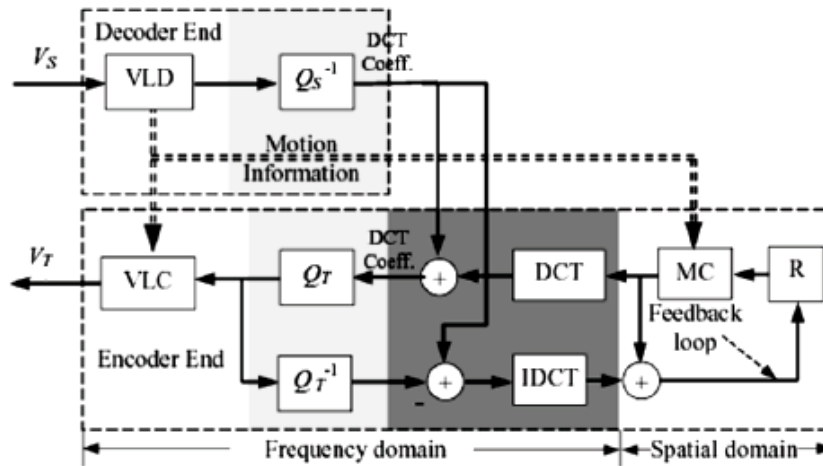


Figure 5.12 Simplified SDTA without STR [29]

5.2.4 Frequency domain transcoding architecture

Some operations in the decoder part of figure 5.11 can be reduced for a simplified SDTA shown in figure 5.12. Further simplification can be achieved if the DCT/IDCT operations can be removed in the encoder by performing motion compensation in frequency domain. Figure 5.13 describes simplified frequency domain transcoding architecture (FDTA) which can

achieve this. Here motion compensation (MC) is achieved in frequency domain. Frequency domain motion compensation techniques are described in [54]. FDTA gives significant reduction in complexity over the cascaded scheme but it can have drift as sub-pixel motion compensation cannot be done at frequency level [29].

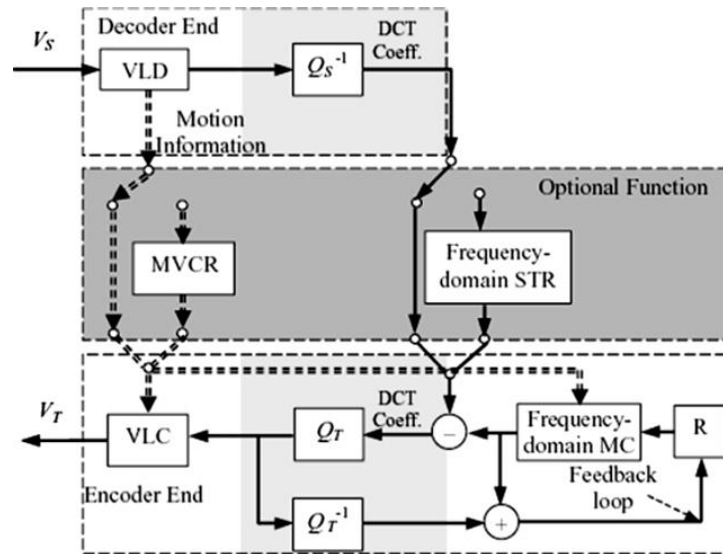


Figure 5.13 Frequency domain transcoding architecture (FDTA) [29]

5.2.5 Spatial resolution reduction

Spatial resolution reduction (SRR) can be achieved using either SDTA or FDTA described in sections 5.2.3 and 5.2.4 respectively. Spatial resolution reduction comes along with bitrate reduction. With the spatial resolution reduction motion vectors need to be recalculated; section 5.2.5.4 describes techniques for motion vector composition and refinement. Also with spatial resolution reduction, the MB mode decisions need to be made again; section 5.2.5.5 discusses MB mode decisions in such scenarios. Sections 5.2.5.1 thru 5.2.5.3 describe different techniques for SRR.

5.2.5.1 Filtering and subsampling

Subsampling is a method to reduce spatial resolution in the spatial domain. Subsampling needs a decimation filter before dropping the alternate pixels. In [55] the authors have proposed a 7-step decimation filter with coefficients as $(-1, 0, 9, 16, 9, 0, -1)/32$ which. The same filter is applied in both horizontal and vertical directions for luminance and chrominance after which downsampling is performed by dropping alternate pixels as shown in figure 5.14.

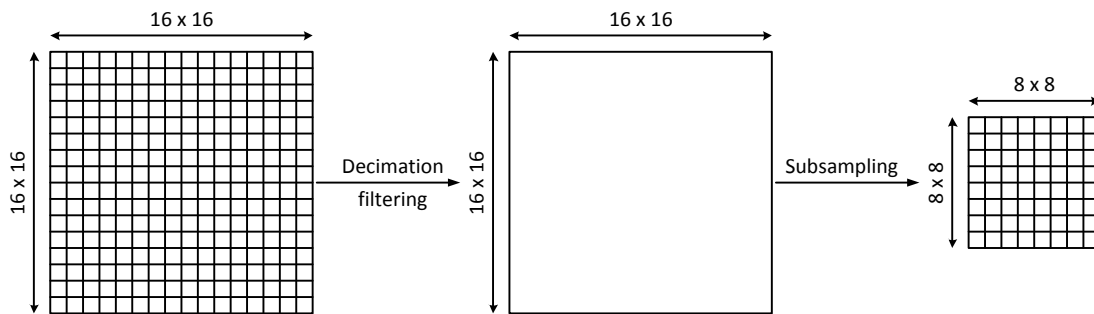


Figure 5.14 Decimation of a 16 x 16 MB

5.2.5.2 Pixel averaging

The simplest method to reduce spatial resolution is pixel averaging as show in figure 5.15. Here every $M \times M$ pixels are represented by 1 pixel in order to achieve $M:1$ spatial resolution reduction. Most often the calculated value for this replacement pixel is the average of the $M \times M$ original pixels. However this can clearly introduce blur in the picture [55].

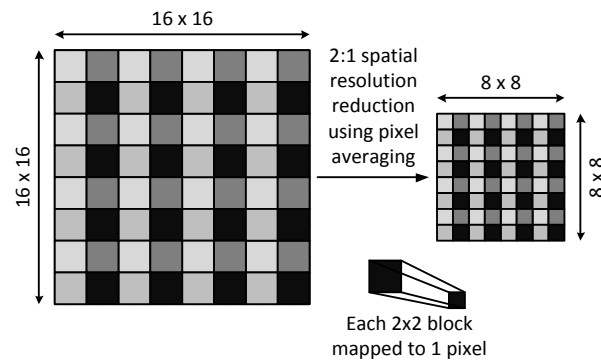


Figure 5.15 Pixel averaging

5.2.5.3 Discarding high frequency DCT coefficients

A more efficient method to achieve spatial resolution reduction is discarding the higher order DCT coefficients. This method performs the scaling operation in DCT domain so it can be used in the FDTA where DCT domain MC is performed. Also it is much simpler as it simply involves discarding all the higher order frequency coefficients other than the lower frequency 8x8 coefficients out of the 16x16 coefficients [55] as shown in figure 5.16. According to a comparative study carried out by the authors in [55] DCT domain decimation technique performs the best of the three SRR techniques described.

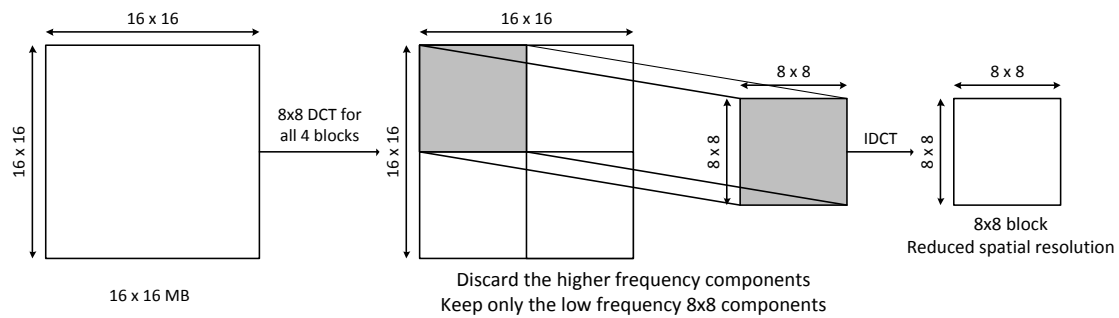


Figure 5.16 DCT domain decimation for SRR

5.2.5.4 Motion vector composition and refinement

For SRR the motion vectors from the source cannot be passed directly to the encoder. When the resolution is reduced by 2:1 as shown in figure 5.17, multiple MVs need to be merged to a single MV. This process is motion vector composition.

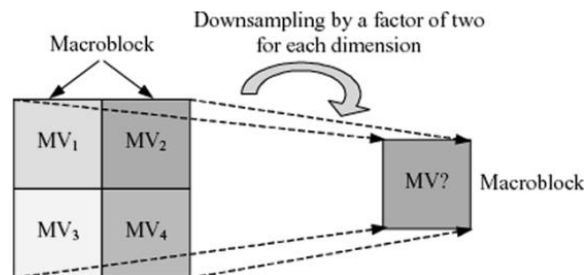
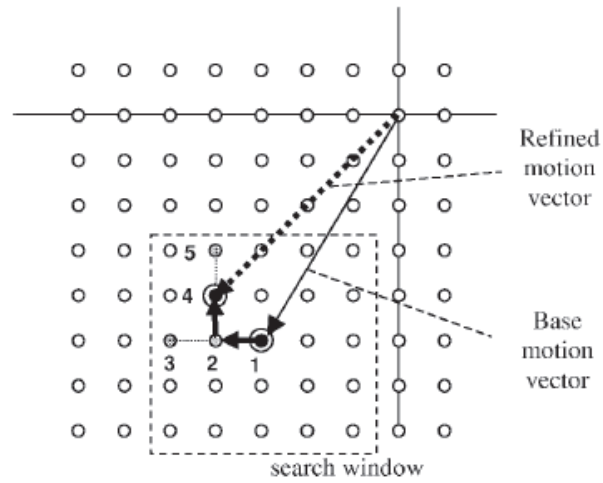


Figure 5.17 Four motion vectors being downsampled to one [29]

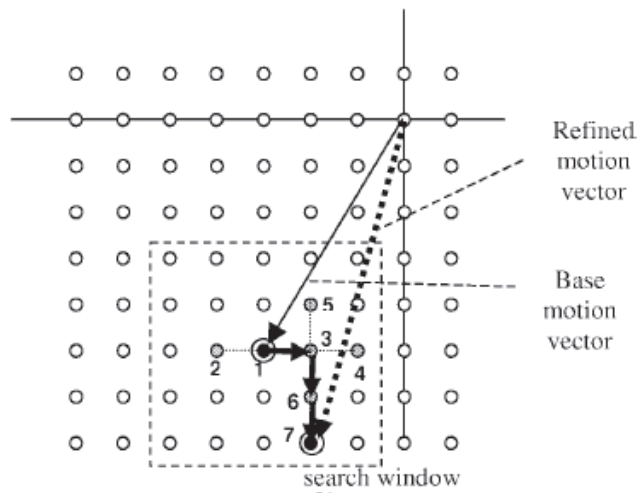
Numerous simple and involved methods have been proposed. Some of them are as follows.

- Random selection: A fast method of MV composition is to select any random MV from the incoming MVs to replace all of them. However this is a very inefficient method. This method is studied in [56].
- Mean: An average value of all the incoming motion vectors can be used; this method however can be used only if all the motion vectors are in the same direction [57]. Most often there is no significant difference between the MV values of neighboring MBs. But this method ineffective if one of the original MVs is much larger compared to the rest of the MVs.
- DCMax: In [58] a technique used for MV composition makes use of the incoming MV with the maximum DC coefficients of the residual block in the source video. This method according to [29] is more complicated but gives better results compared to taking mean or random selection.

The motion vector composition schemes are sub-optimal and introduce degradation in quality at the output. So MV refinement techniques are proposed in [55] [56]. The recalculated MVs will not differ very much from actual motion vectors; so refinement of this MV in a small search window gives better results. Figure 5.18 describes a MV refinement scheme proposed in [51].



(a)



(b)

Figure 5.18 Motion vector refinement using search window (a) best case – small search (b) worst case – long search [51]

5.2.5.5 MB coding mode decision

With the reduced spatial resolution the MB coding modes need to be reevaluated. For 2:1 downscaling four incoming MBs are coded to a single MB. There can be two scenarios – all four of them have the same MB coding mode or all four of them have a different coding mode

as shown in figure 5.19. In [56] the following scheme is proposed when all four incoming MBs have the same coding mode.

- If incoming are coded are INTRA MBs re-encode the reduced MB as INTRA.
- If the incoming MBs are SKIPPED again code the reduced MB as SKIPPED.
- If the incoming MBs are coded as INTER, check to see if all the coefficients in the reduced MB are zero; than it should be coded as SKIPPED. If not it should be encoded as either INTRA or INTER.

For the second scenario as shown in figure 5.19, authors in [59] have described the following scheme.

- At the transmitter: If one of the four is an INTRA coded MB, than pass the new MB as intra. If one of them is INTER and none of them INTRA, pass the new MB as INTER MB. If all MBs are SKIPPED pass the new MB as SKIPPED.
- At the encoder: Re-evaluate the MBs as applicable.

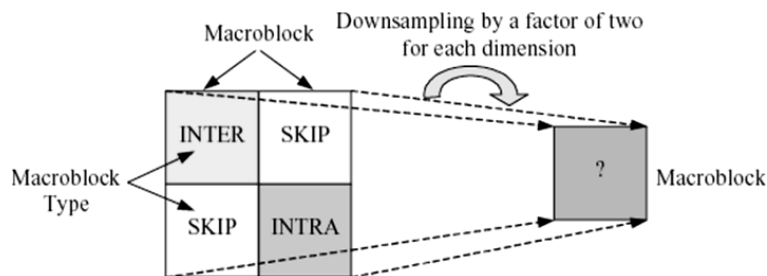


Figure 5.19 Four macroblock types downsampled to one [29]

5.2.6 Temporal resolution reduction

With frames dropped in order to achieve temporal resolution reduction (TRR) the incoming MVs are not valid as they point to the frames that do not exist in the transcoded bitstream. New MVs need to be derived. Sections 5.2.5.1 thru 5.2.5.4 describe different technique to derive new MVs from the MVs of the dropped frames.

5.2.6.1 Bilinear Interpolation

In [60] a bilinear interpolation method to estimate the new MVs is proposed. The new MV is calculated using interpolation of MVs between every adjacent frame between the current frame and previous non-skipped frame. The new location position based on this interpolated MV serves as the search center to calculate the actual value of the new MV; thereby reducing the complexity in the new MV search. The search area is calculated from the number of skipped frames and accumulated magnitudes of their MVs.

5.2.6.2 Forward Dominant Vector Selection (FDVS)

The method proposed by the authors in [51] selects the dominant MV from the four neighboring macroblocks as shown in figure 5.20. A dominant MV is defined as the MV carried by the macroblock that has the largest overlapping segment with the block pointed by the incoming MV. The best-matched area pointed by the MV of the current macroblock occurring after a dropped frame overlaps with at most four macroblocks in the previous dropped frame. The MV of the macroblock with the largest overlapping portion is selected and added to the current MV. This process is repeated each time a frame is dropped until a new set of MVs is composed for the last non skipped frame.

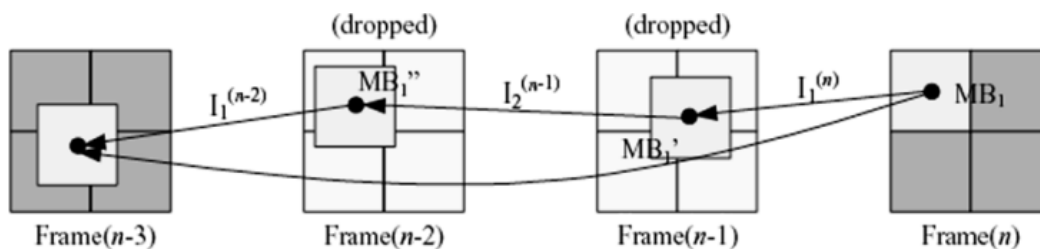


Figure 5.20 FDVS motion vector composition scheme for TRR [29]

5.2.6.3 Telescopic Vector Composition (TVC)

The TVC technique is described in [55]. It accumulates all the MVs of the corresponding macroblocks of the dropped frames and adds each resultant composed MV to the corresponding MV in the current frame. This technique also carries out new macroblock decision and MV refinement.

5.2.6.4 Activity-Dominant Vector Selection (ADVS)

The authors in [58] describe this technique which makes use of the activity of the macroblock to choose the new MV. The activity information of a macroblock is represented by counting the number of nonzero quantized DCT coefficients of covered 8 x 8 residual blocks; other statistics, such as the sum of the absolute values of DCT coefficients, etc. These quantities are proportional to the spatial-activity measurement. The higher the activity of the macroblock, the more significant will be the motion of the macroblock. Since the quantized DCT coefficients of prediction errors are available in the incoming bitstream of transcoder, the computation for counting the nonzero coefficients is very little.

CHAPTER 6

IMPLEMENTATION, RESULTS AND CONCLUSIONS

6.1 Cascaded decoder and encoder

The simplest implementation of a transcoder as described in chapter 5 is to cascade the decoder and encoder to get the new bitstream. As it involves complete decoding and re-encoding the complexity of such an implementation is very high. However the only error it has is from lossy encoding of already degraded video output for the decoder. This implementation is devoid of drift errors (section 5.2.2.1). Being the simplest implementation of a transcoder, this architecture is used for the basis of complexity and output quality comparison. In the current research the first step is the implementation of cascaded transcoding as described in figure 6.1.

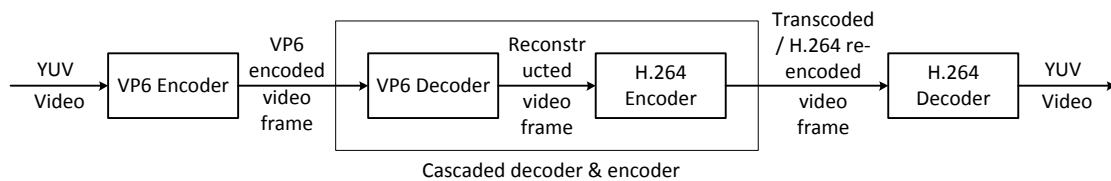


Figure 6.1 Cascade decoder and encoder

Three video sequences – 2 QCIF video sequences (Akiyo [55] and Foreman [56]) and 1 CIF sequence (Stefan [55]), same as used in section 4.4 - are used for the output quality metrics. Tables 6.1 thru 6.3 give the quality metrics – mean squared error (MSE), peak signal to noise ratio (PSNR) and structural similarity index metric (SSIM) [54] for the Y, U and V components averaged over the video sequences. Figures 6.2, 6.6, 6.7, 6.11 and 6.12 show selected frames from the video sequences. The graphs in figures 6.3 thru 6.5, figures 6.8 thru 6.10 and figures 6.13 thru 6.15 give a comparison of the Y component MSE, PSNR and SSIM for the cascaded transcoder with that of VP6 and H.264 codecs for all the three clips.

Table 6.1 Foreman sequence – quality metrics for cascaded implementation (PSNR in dB)

Original (VP6) bitrate (kbps)	Transcoded (H.264) bitrate (kbps)	Metrics type	Original (VP6) metrics	H.264 direct encoding metrics	Transcoded output metric wrt VP6	Transcoded output metrics wrt original
9.324	9.220	Y MSE	25.640	26.396	20.621	38.579
		U MSE	9.998	6.330	6.221	12.964
		V MSE	6.816	5.089	2.671	9.022
		Y PSNR	34.099	34.404	35.339	32.479
		U PSNR	38.152	40.138	40.279	37.007
		V PSNR	39.798	41.068	43.868	38.579
		Y SSIM	0.921	0.946	0.942	0.918
		U SSIM	0.958	0.970	0.985	0.953
		V SSIM	0.951	0.960	0.984	0.941
10.648	9.800	Y MSE	24.292	19.242	15.014	31.718
		U MSE	9.646	6.056	5.736	12.212
		V MSE	6.504	5.004	2.495	8.185
		Y PSNR	34.329	35.459	36.433	33.182
		U PSNR	38.302	40.315	40.574	37.264
		V PSNR	40.003	41.139	44.162	39.001
		Y SSIM	0.923	0.953	0.947	0.925
		U SSIM	0.959	0.970	0.986	0.955
		V SSIM	0.952	0.960	0.983	0.945
14.536	13.910	Y MSE	14.354	12.876	11.800	21.505
		U MSE	5.866	5.837	4.947	10.617
		V MSE	3.274	4.923	2.933	7.008
		Y PSNR	36.576	37.040	37.413	34.832
		U PSNR	40.456	40.469	41.188	37.877
		V PSNR	42.983	41.211	43.462	39.675
		Y SSIM	0.964	0.961	0.955	0.940
		U SSIM	0.984	0.970	0.984	0.959
		V SSIM	0.981	0.960	0.980	0.950
53.309	70.030	Y MSE	4.367	3.140	3.449	
		U MSE	1.937	2.218	1.830	
		V MSE	1.536	1.798	1.499	
		Y PSNR	41.758	43.193	42.778	
		U PSNR	45.275	44.685	45.522	
		V PSNR	46.281	45.594	46.389	
		Y SSIM	0.977	0.985	0.983	
		U SSIM	0.986	0.984	0.991	
		V SSIM	0.985	0.983	0.988	

Foreman - original
Frame 1 - I frame



(a)

Foreman - VP6
PSNR - 34.14 dB
SSIM - 0.9260



(c)

Foreman - cascaded transoder
PSNR - 32.9 dB
SSIM - 0.9195



(b)

Foreman - H.264 Baseline
PSNR - 35.82 dB
SSIM - 0.9474



(d)

Figure 6.2 Foreman sequence (bitrate ~ 10 kbps) – frame 1 (a) original sequence (b) cascaded decoder and encoder (c) VP6 (d) H.264 baseline

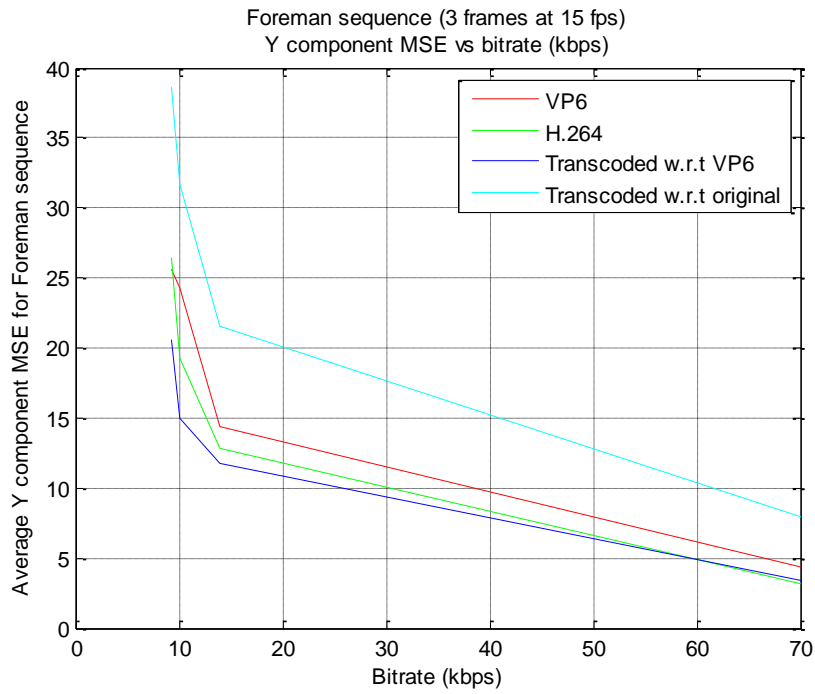


Figure 6.3 Foreman sequence - Y component MSE - cascaded implementation

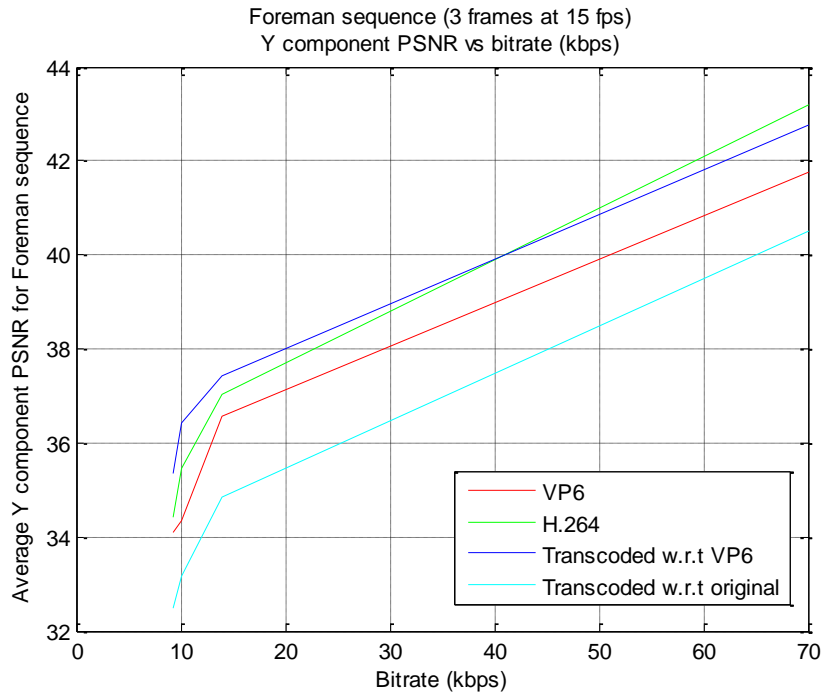


Figure 6.4 Foreman sequence – Y component PSNR (dB) – cascaded implementation

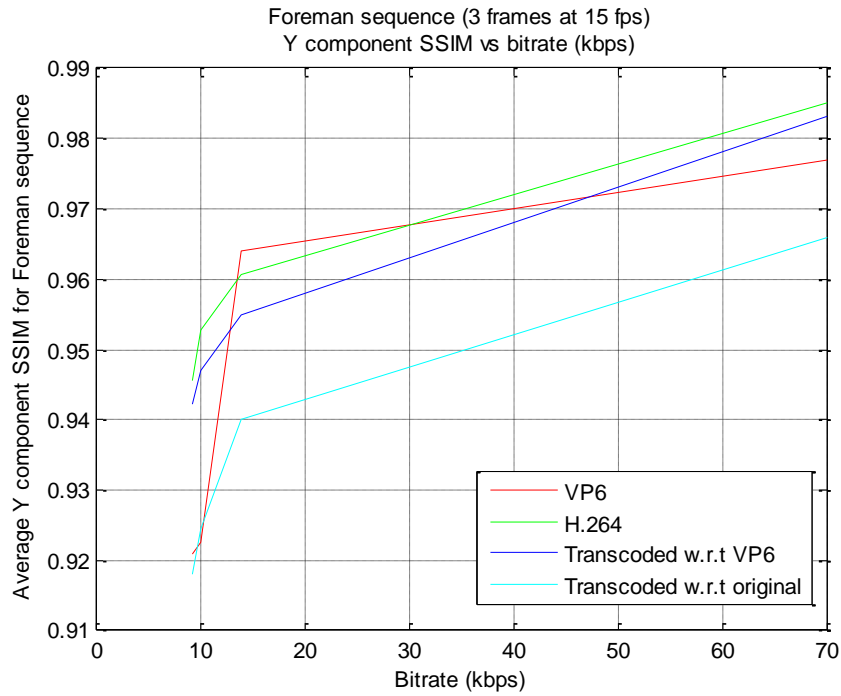


Figure 6.5 Foreman sequence – Y component SSIM – cascaded implementation

Table 6.2 Akiyo sequence – quality metrics for cascaded implementation (PSNR in dB)

Original (VP6) bitrate (kbps)	Original H.264 bitrate (kbps)	Transcoded (H.264) bitrate (kbps)	Metrics type	Original (VP6) metrics	H.264 direct encoding metrics	Transcoded output metric wrt VP6	Transcoded output metrics wrt original
85.781	94.17	93.047	Y MSE	33.898	17.044	25.425	50.863
			U MSE	7.067	5.599	1.675	8.555
			V MSE	5.808	3.620	2.018	7.053
			Y PSNR	32.923	31.418	34.345	31.245
			U PSNR	39.667	39.276	45.949	38.825
			V PSNR	40.515	40.622	45.113	39.664
			Y SSIM	0.908	0.891	0.936	0.890
			U SSIM	0.945	0.944	0.989	0.940
			V SSIM	0.964	0.960	0.989	0.958
138.008	138.086	120.625	Y MSE	26.658	17.044	15.715	34.319
			U MSE	6.380	5.599	1.723	7.761
			V MSE	4.948	3.620	2.042	6.792
			Y PSNR	33.992	35.210	36.171	32.820
			U PSNR	40.110	40.199	45.772	39.238
			V PSNR	41.209	41.409	45.035	39.814
			Y SSIM	0.922	0.940	0.954	0.915
			U SSIM	0.946	0.950	0.988	0.941
			V SSIM	0.966	0.964	0.987	0.958

Table 6.2 - *Continued*

Original (VP6) bitrate (kbps)	Original H.264 bitrate (kbps)	Transcoded (H.264) bitrate (kbps)	Metrics type	Original (VP6) metrics	H.264 direct encoding metrics	Transcoded output metric wrt VP6	Transcoded output metrics wrt original
167.148	160.195	144.687	Y MSE	19.695	56.204	51.001	66.206
			U MSE	5.527	4.885	1.571	6.846
			V MSE	4.166	3.500	1.800	5.919
			Y PSNR	35.284	32.841	33.229	31.065
			U PSNR	40.720	41.460	46.624	39.827
			V PSNR	41.953	42.974	46.062	40.475
			Y SSIM	0.935	0.904	0.912	0.881
			U SSIM	0.952	0.956	0.988	0.944
			V SSIM	0.969	0.973	0.989	0.962
361.992	360.787	350.273	Y MSE	6.980	5.180	5.321	9.707
			U MSE	2.643	2.918	1.689	4.151
			V MSE	1.747	1.894	1.357	2.713
			Y PSNR	39.832	41.151	40.995	38.261
			U PSNR	43.960	43.525	45.932	41.953
			V PSNR	45.773	45.417	46.878	43.803
			Y SSIM	0.969	0.977	0.975	0.964
			U SSIM	0.972	0.969	0.983	0.959
			V SSIM	0.983	0.982	0.988	0.976

Akiyo - original
Frame 2 - 1st frame after I-frame



(a)

Akiyo - VP6
PSNR - 38.33 dB
SSIM - 0.9353



(c)

Akiyo - cascade transcoder
PSNR - 37.13 dB
SSIM - 0.9314



(b)

Akiyo - H.264
PSNR - 40.49 dB
SSIM - 0.9621



(d)

Figure 6.6 Akiyo sequence (bitrate ~ 9.3 kbps) – frame 2 (a) original sequence (b) cascaded decoder and encoder (c) VP6 (d) H.264 baseline

Akiyo - original
Frame 30 - Last frame



(a)

Akiyo - VP6
PSNR - 33.36 dB
SSIM - 0.9151



(c)

Akiyo - cascaded transcoder
PSNR - 31.03 dB
SSIM - 0.9042



(b)

Akiyo - H.264
PSNR - 32.17 dB
SSIM - 0.9307



(d)

Figure 6.7 Akiyo sequence (bitrate ~ 9.3 kbps) – frame 2 (a) original sequence (b) cascaded decoder and encoder (c) VP6 (d) H.264 baseline

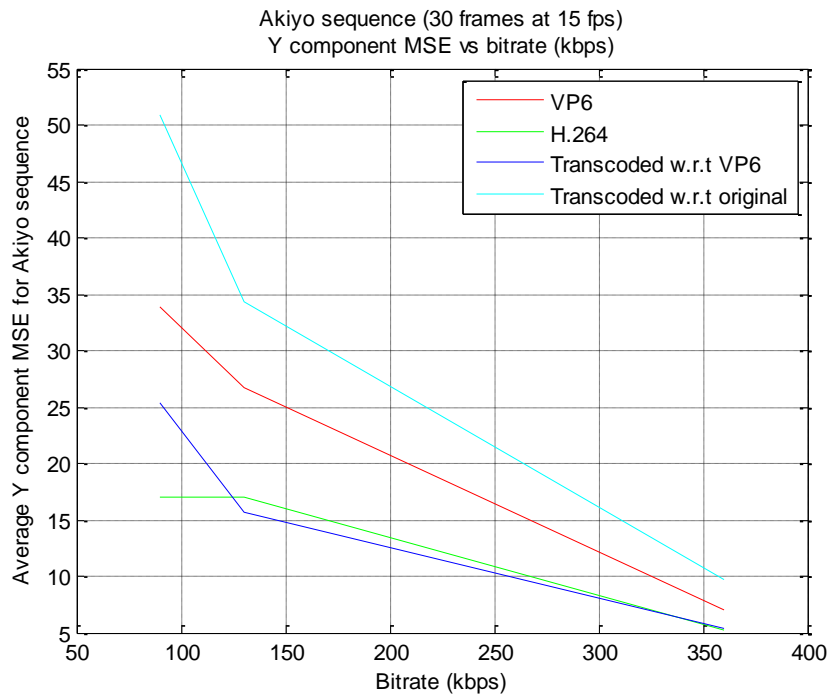


Figure 6.8 Akiyo sequence – Y component MSE – cascaded implementation

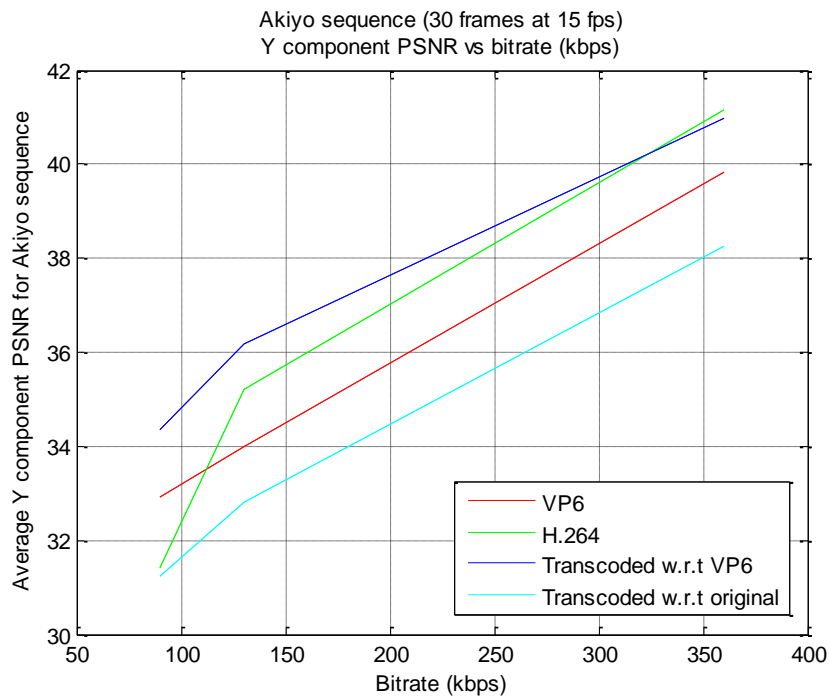


Figure 6.9 Akiyo sequence – Y component PSNR – cascaded implementation

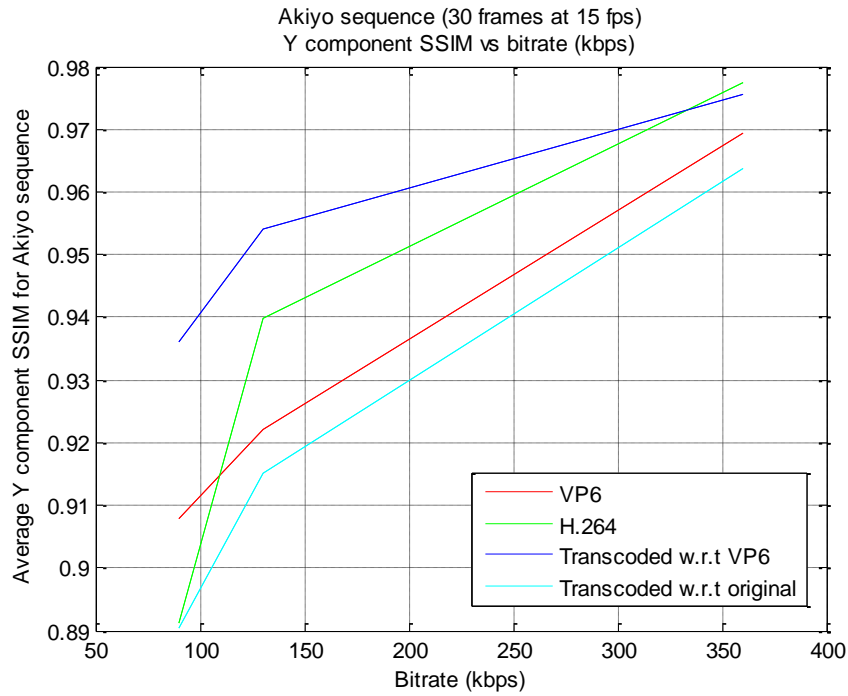


Figure 6.10 Akiyo sequence – Y component SSIM – cascaded implementation

Table 6.3 Stefean sequence – quality metrics for cascaded implementation (PSNR in dB)

Original (VP6) bitrate (kbps)	Original H.264 bitrate (kbps)	Transcod ed (H.264) bitrate (kbps)	Metrics type	Original (VP6) metrics	H.264 direct encoding metrics	Transcode d output metric wrt VP6	Transcod ed output metrics wrt original
5441.656	5514.090	5541.100	Y MSE	0.521	0.871	0.863	1.320
			U MSE	0.495	0.764	0.743	1.117
			V MSE	0.484	0.761	0.738	1.102
			Y PSNR	50.965	50.937	50.987	48.008
			U PSNR	51.185	50.620	50.754	48.347
			V PSNR	51.279	50.698	50.885	48.473
			Y SSIM	0.997	0.997	0.997	0.995
			U SSIM	0.995	0.993	0.994	0.990
			V SSIM	0.995	0.994	0.994	0.991
1209.406	1242.820	1247.010	Y MSE	9.300	10.167	10.467	16.138
			U MSE	4.568	6.504	5.022	9.683
			V MSE	4.543	6.439	5.062	9.760
			Y PSNR	38.504	38.911	38.707	36.426
			U PSNR	41.593	40.199	41.422	38.353
			V PSNR	41.621	40.261	41.374	38.313
			Y SSIM	0.977	0.982	0.979	0.975
			U SSIM	0.965	0.954	0.965	0.931
			V SSIM	0.969	0.958	0.966	0.936

Table 6.3 – Continued

Original (VP6) bitrate (kbps)	Original H.264 bitrate (kbps)	Transcoded (H.264) bitrate (kbps)	Metrics type	Original (VP6) metrics	H.264 direct encoding metrics	Transcoded output metric wrt VP6	Transcoded output metrics wrt original
660.862	611.710	609.090	Y MSE	21.940	20.647	20.620	32.461
			U MSE	8.498	10.454	6.098	14.751
			V MSE	8.634	10.416	6.244	15.385
			Y PSNR	34.842	34.990	34.992	33.029
			U PSNR	38.918	37.944	40.295	36.458
			V PSNR	38.854	37.957	40.185	36.274
			Y SSIM	0.962	0.971	0.967	0.959
			U SSIM	0.939	0.927	0.959	0.899
254.650	221.060	201.120	V SSIM	0.945	0.935	0.960	0.905
			Y MSE	65.050	74.135	67.532	102.959
			U MSE	16.778	16.532	6.746	22.354
			V MSE	17.708	17.608	7.228	24.240
			Y PSNR	30.162	29.695	30.117	28.221
			U PSNR	35.968	36.049	39.900	34.696
			V PSNR	35.752	35.786	39.588	34.341
			Y SSIM	0.918	0.927	0.921	0.900
180.822	160.520	150.800	U SSIM	0.889	0.899	0.963	0.866
			V SSIM	0.893	0.902	0.961	0.866
			Y MSE	82.706	214.934	205.673	260.859
			U MSE	18.978	19.323	9.433	24.661
			V MSE	19.872	20.370	10.735	27.151
			Y PSNR	29.113	25.516	25.778	24.572
			U PSNR	35.384	35.416	38.557	34.289
			V PSNR	35.193	35.206	37.986	33.877
Y SSIM	0.908	0.839	0.827	0.802			
U SSIM	0.881	0.889	0.951	0.861			
V SSIM	0.888	0.893	0.946	0.860			

Stefan - original
Frame 1 - I-frame



(a)

Stefan - cascaded transcoder
PSNR - 39.50 dB
SSIM - 0.9848



(b)

Stefan - VP6
PSNR - 51.02 dB
SSIM - 0.9972



(c)

Stefan - H.264
PSNR - 39.7 dB
SSIM - 0.9852



(d)

Figure 6.11 Stefan sequence – frame 1 – I frame (a) original sequence (b) cascaded decoder and encoder (c) VP6 (d) H.264 baseline

Stefan - original
Frame 15



(a)

Stefan - cascaded transcoder
PSNR - 48.60 dB
SSIM - 0.9954



(b)

Stefan - VP6
PSNR - 50.94 dB
SSIM - 0.9972



(c)

Stefan - H.264
PSNR - 51.82 dB
SSIM - 0.9977



(d)

Figure 6.12 Stefan sequence – frame 15 – an intermediate frame (a) original sequence (b) cascaded decoder and encoder (c) VP6 (d) H.264 baseline

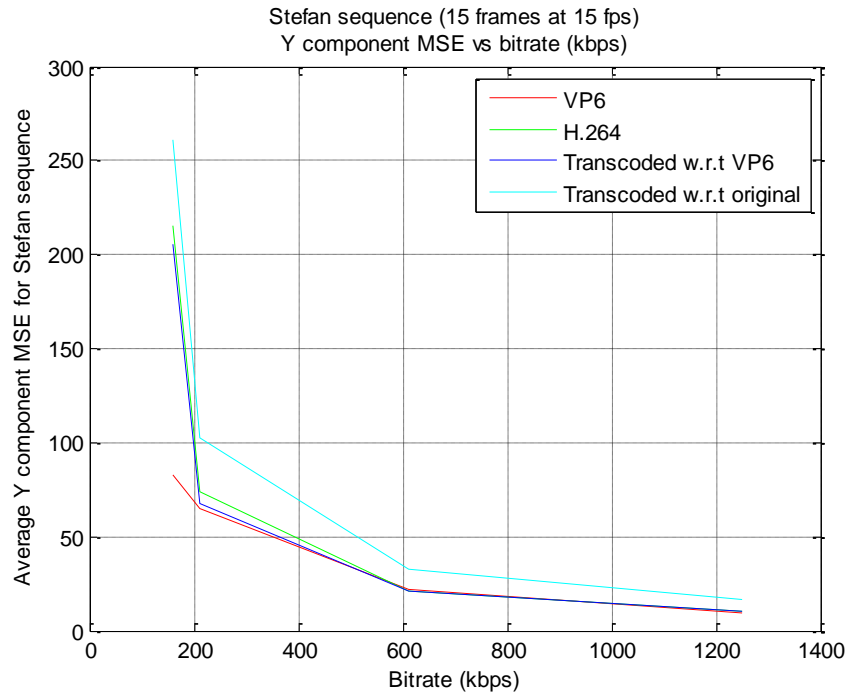


Figure 6.13 Stefan sequence – Y component MSE – cascaded implementation

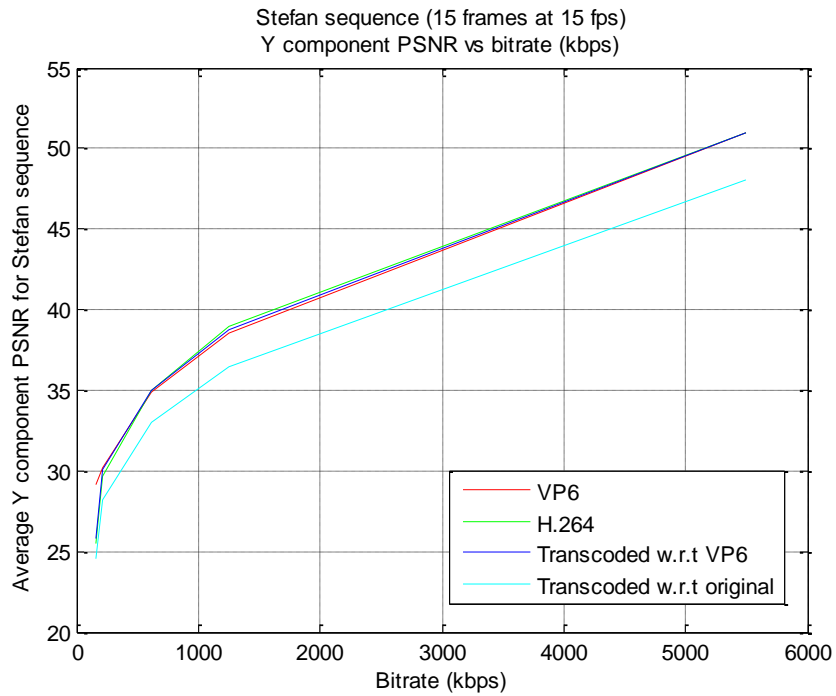


Figure 6.14 Stefan sequence – Y component PSNR (dB) – cascaded implementation

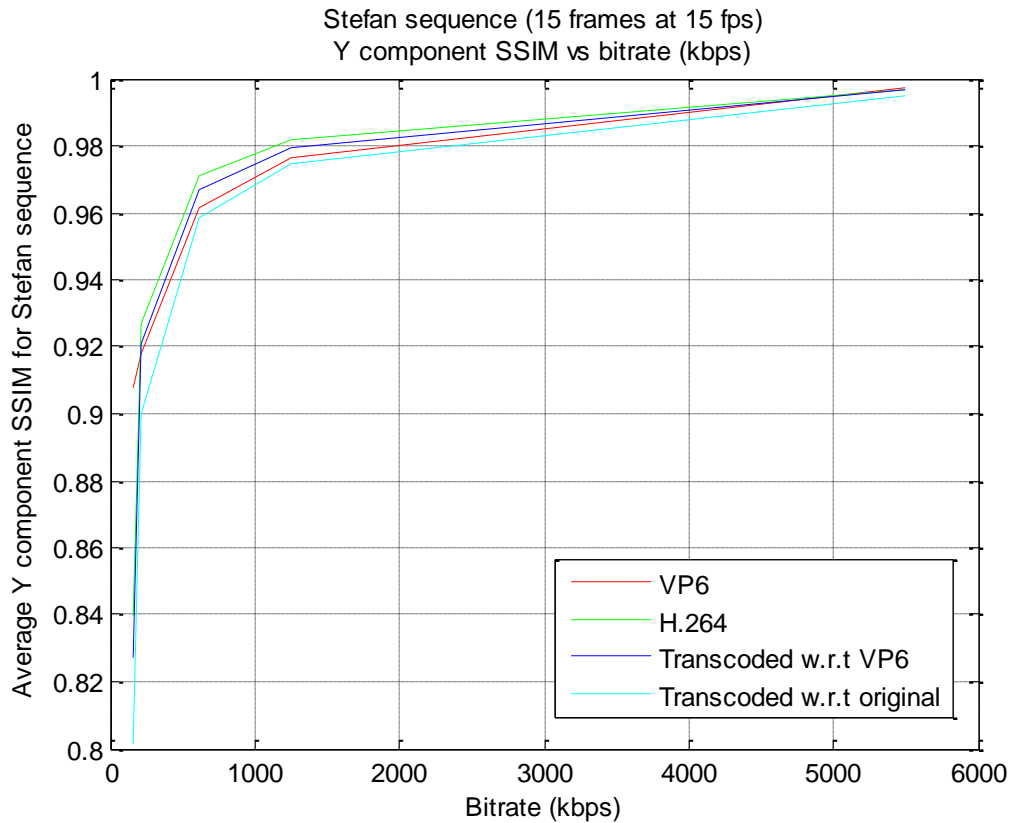


Figure 6.15 Stefan sequence – Y component SSIM – cascaded implementation

6.2 Proposed technique – reuse of motion information from VP6

In the encoding process, up to 70% of the time complexity comes from motion estimation (ME) [30]. H.264 is a complex codec and the motion estimation process in H.264 is highly involved. If the motion estimation time (MET) in the encoding process of a predicted frame can be reduced, the complexity of the transcoder is highly reduced. This is the basis of the proposed technique.

When the input VP6 file is decoded, motion information is available as VP6 also uses motion estimation for temporal redundancy reduction. The motion estimation techniques in VP6 are less involved compared to H.264. Table 6.4 gives a comparison of motion estimation process in both the codecs.

Table 6.4 Motion estimation and compensation (VP6 vs. H.264)

	VP6	H.264
Motion vector resolution	¼ pixels	¼ pixels
Number of reference frames	1 previous & 1 golden	Up to 16 reference frames
Block sizes	4x4, 4x8, 8x4, 8x8, 8x16, 16x8 and 16x16	8x8 and 16x16
Maximum motion vector search range	16 pixels	32 pixels
Use of golden frame?	Yes	No
Bidirectional prediction?	No	Yes (not in baseline profile)

As it can be seen both the codecs support up to quarter-pixel accuracy for motion vector prediction. If the motion vectors of VP6 can be used in encoding to H.264, the time in calculating a new H.264 motion vector for the corresponding MB can be saved. However in VP6, motion estimation block size can only be 16 x 16 or 8 x 8 based on selected mode (see section 3.6) as compared to a range of MB and sub-MB sizes available for H.264 (section 2.4). So in the current technique the motion vectors of VP6 are used and the block size in encoding H.264 with these motion vectors is restricted to 16 x 16 and 8 x 8 depending on the input VP6 MB mode. Unlike H.264, VP6 does not support multiple frame prediction but it has a special golden frame buffer used in motion prediction. So H.264 encoding in the proposed transcoding technique is restricted to only previous frame prediction. According the [37], the frequency of golden frame prediction usage is only 11%. So whenever the input vector is a golden frame vector a new motion vector is recalculated using the previous frame for prediction. Figure 6.16 gives an overview of proposed technique. Tables 6.5 thru 6.7 give a comparison of the output quality and the motion estimation times when a sequence is transcoded using cascaded architecture and the proposed technique.

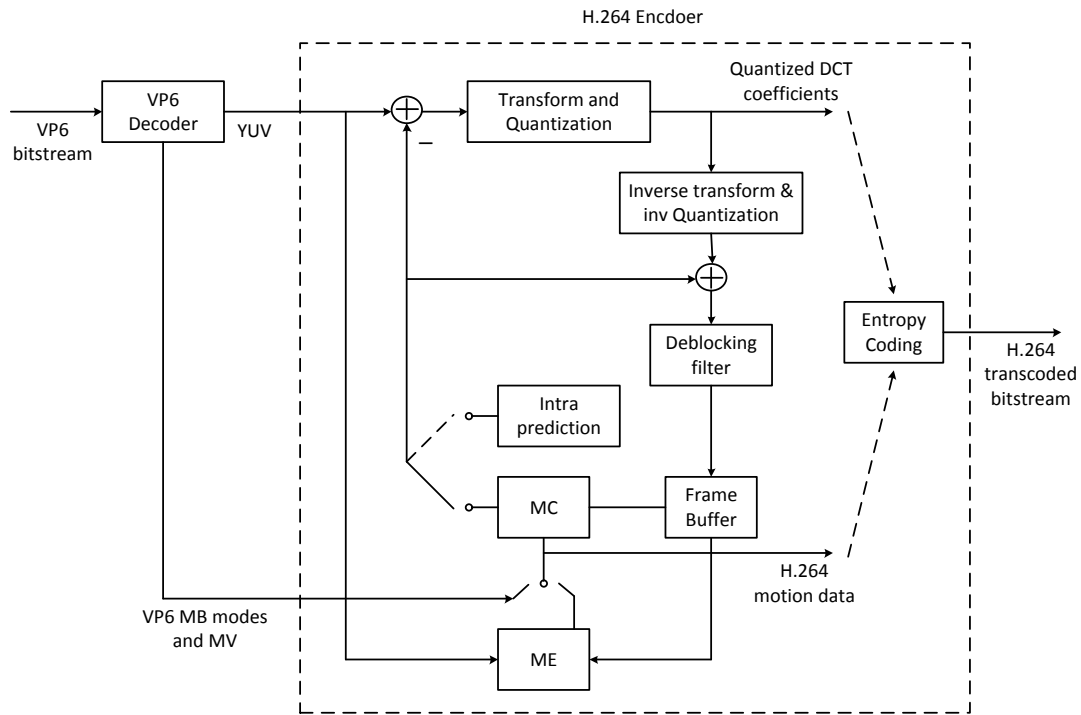


Figure 6.16 Proposed technique of reusing MB modes and MV from VP6

Table 6.5 PSNR (dB) and MET for Foreman clip using cascaded architecture and the proposed technique

VP6 bitrate (kbps)	Cascaded				Proposed technique			
	Bitrate (kbps)	PSNR w.r.t VP6 decoded file	PSNR w.r.t original file	MET (motion estimation time) (ms)	Bitrate (kbps)	PSNR w.r.t VP6 decoded file	PSNR w.r.t original file	MET (motion estimation time) (ms)
1096	951.48	30.922	27.1753	90717	946.52	30.848	27.1677	9321
		31.23	27.173	97303		31.188	27.156	9347
1352	1357	33.292		44612	1332	33.196	28.719	8837
		33.78		88961		33.737	28.579	8912
1872	1843.52	35.733	31.2787	43852	1816.64	35.66	31.3079	8746
		36.345	31.308	87619		36.23	31.2919	9535
2488	2560.7	39.198	34.013	45556	2511.48	39.104	33.9561	9460
		39.931	33.998	92706		39.853	33.9887	9181

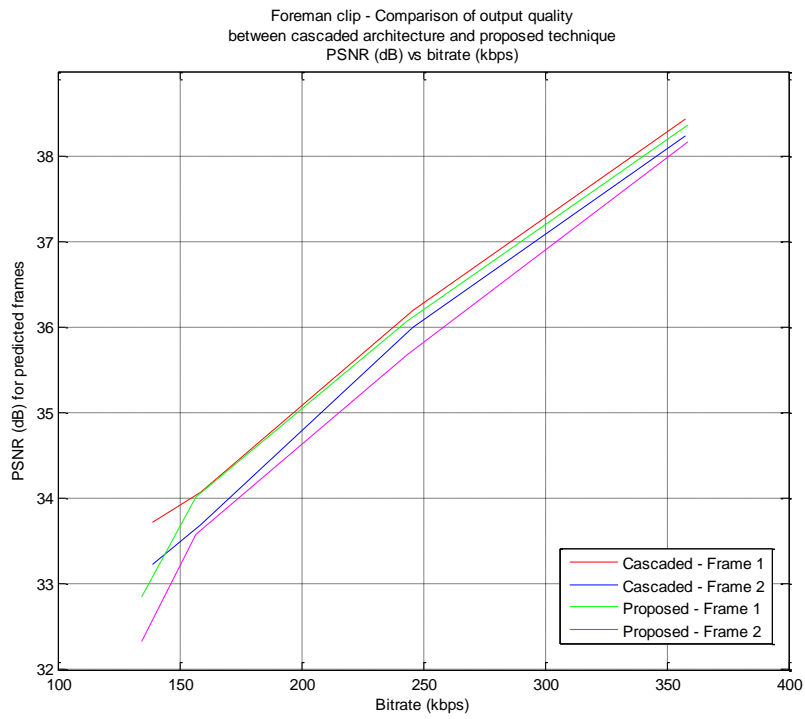
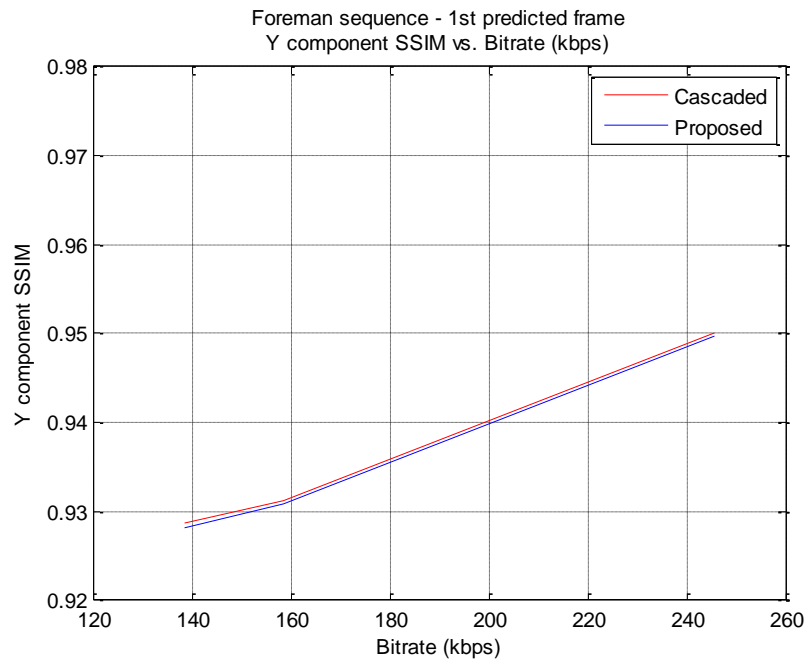
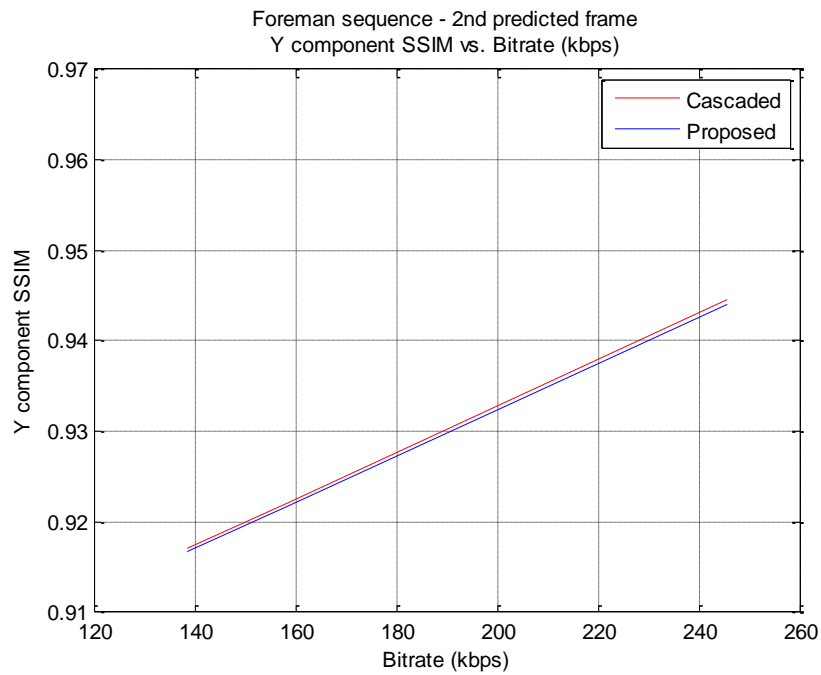


Figure 6.17 Foreman clip – PSNR (dB) vs bitrate (kbps) – Cascaded architecture and proposed technique



(a)



(b)

Figure 6.18 (a) Foreman sequence – Y component SSIM for predicted frame 1
(b) Foreman sequence – Y component SSIM for predicted frame 2

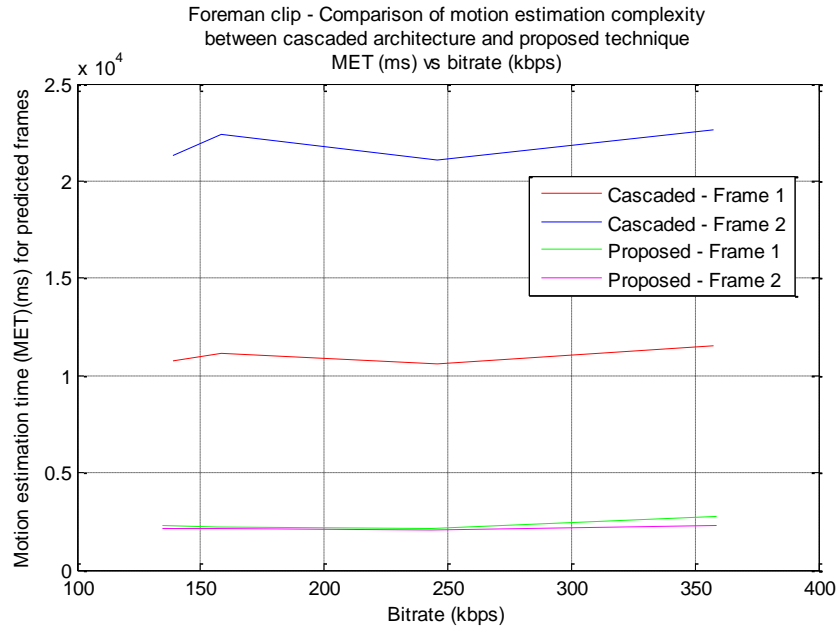


Figure 6.19 Foreman clip – Motion estimation time (MET) vs bitrate (kbps) – Cascaded architecture and proposed technique

Table 6.6 PSNR (dB) and MET for Akiyo clip using cascaded architecture and the proposed technique

VP6 bitrate (kbps)	cascaded				Proposed technique			
	Bitrate (kbps)	PSNR w.r.t VP6 decoded file	PSNR w.r.t original file	MET (motion estimation time) (ms)	Bitrate (kbps)	PSNR w.r.t VP6 decoded file	PSNR w.r.t original file	MET (motion estimation time) (ms)
348.800	357.720	41.178	38.440	11552	358.52	41.018	38.38	2745
		41.445	38.250	22597		41.369	38.17	2274
235.200	245.600	39.349		10590	243.16	39.303	36.078	2146
		39.354		21070		39.257	35.679	2073
162.400	158.480	37.164	34.076	11122	156.64	37.184	34.0218	2248
		37.087	33.704	22420		36.892	33.5798	2104
132.800	138.640	36.534	33.719	10737	134.6	36.486	32.8545	2288
		36.264	33.233	21301		36.386	32.331	2150

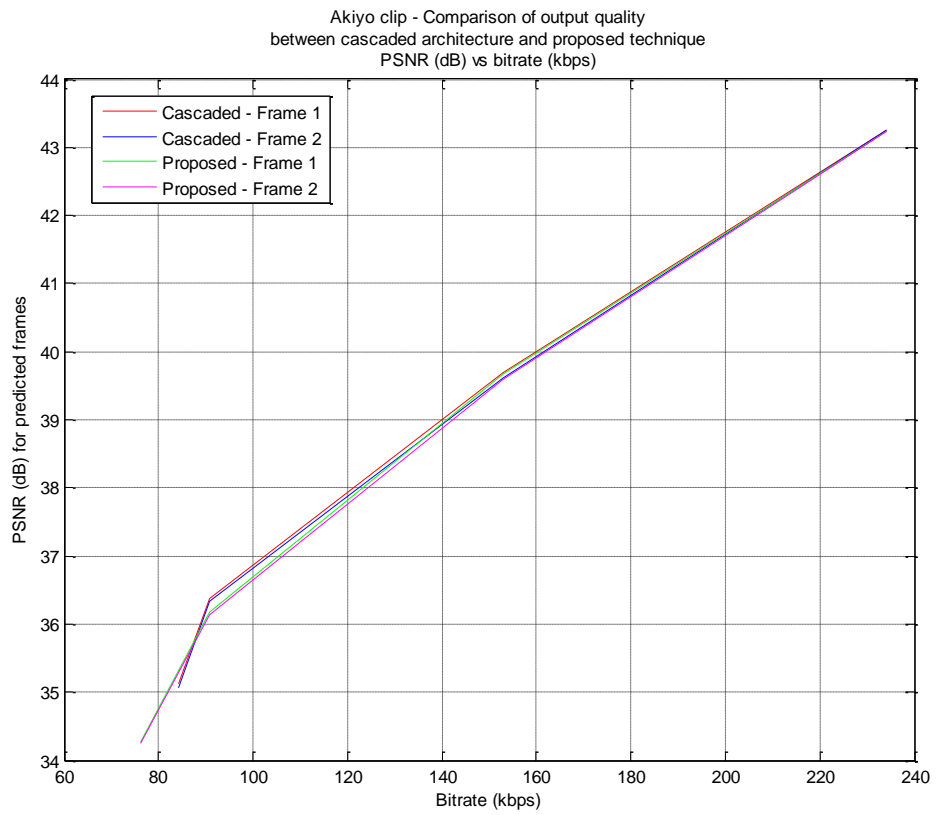


Figure 6.20 Akiyo clip – PSNR (dB) vs bitrate (kbps) – Cascaded architecture and proposed technique

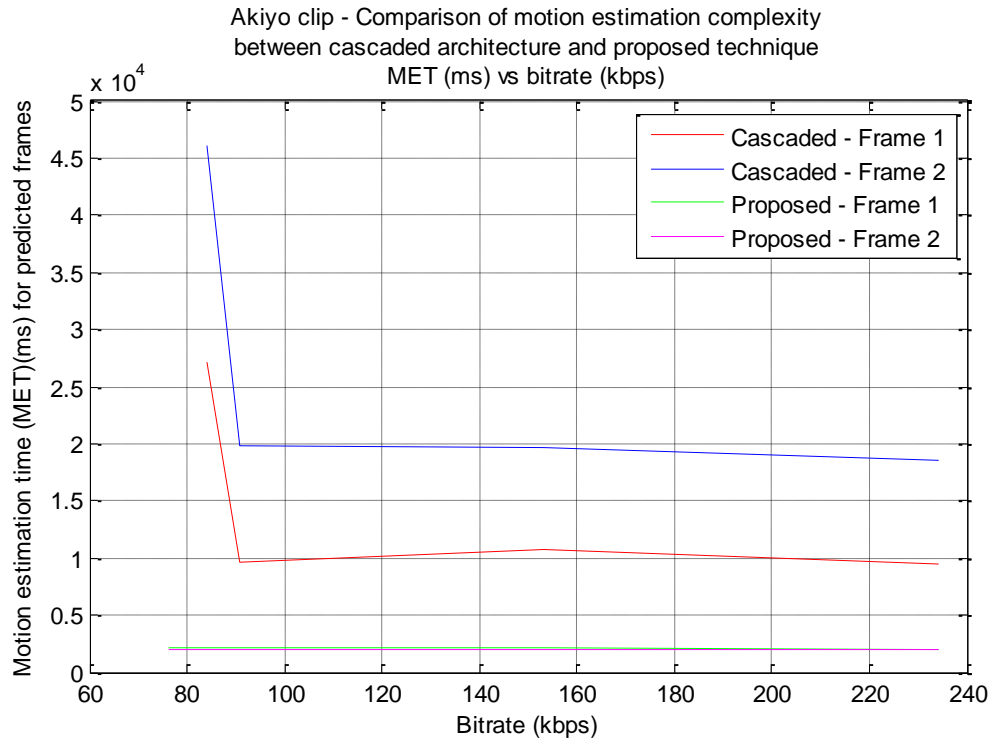


Figure 6.21 Akiyo clip – Motion estimation time (MET) vs bitrate (kbps) – Cascaded architecture and proposed technique

Table 6.7 PSNR (dB) and MET for Stefan clip using cascaded architecture and the proposed technique

VP6 bitrate (kbps)	cascaded				Proposed technique			
	Bitrate (kbps)	PSNR w.r.t VP6 decoded file	PSNR w.r.t original file	MET (motion estimation time) (ms)	Bitrate (kbps)	PSNR w.r.t VP6 decoded file	PSNR w.r.t original file	MET (motion estimation time) (ms)
1096	951.48	30.922	27.1753	90717	946.52	30.848	27.1677	9321
		31.23	27.173	97303		31.188	27.156	9347
1352	1357	33.292		44612	1332	33.196	28.719	8837
		33.78		88961		33.737	28.579	8912
1872	1843.52	35.733	31.2787	43852	1816.64	35.66	31.3079	8746
		36.345	31.308	87619		36.23	31.2919	9535
2488	2560.7	39.198	34.013	45556	2511.48	39.104	33.9561	9460
		39.931	33.998	92706		39.853	33.9887	9181

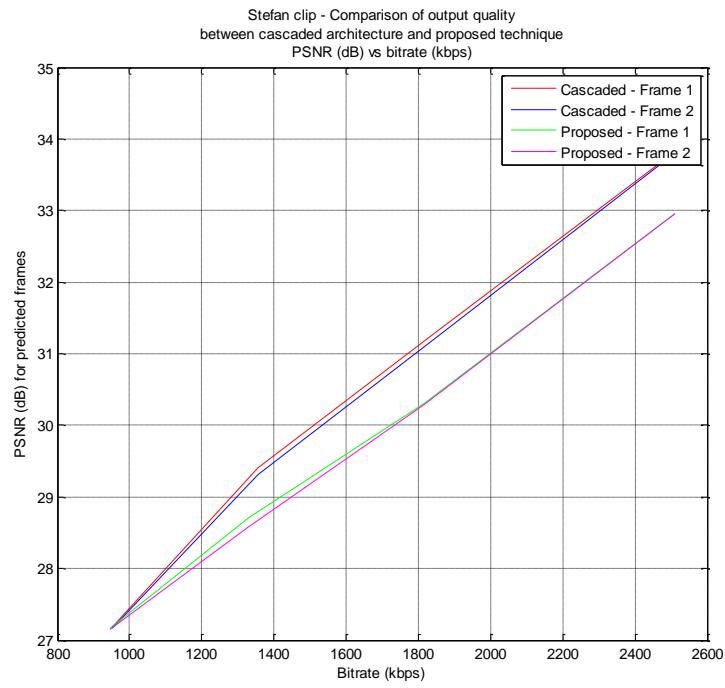
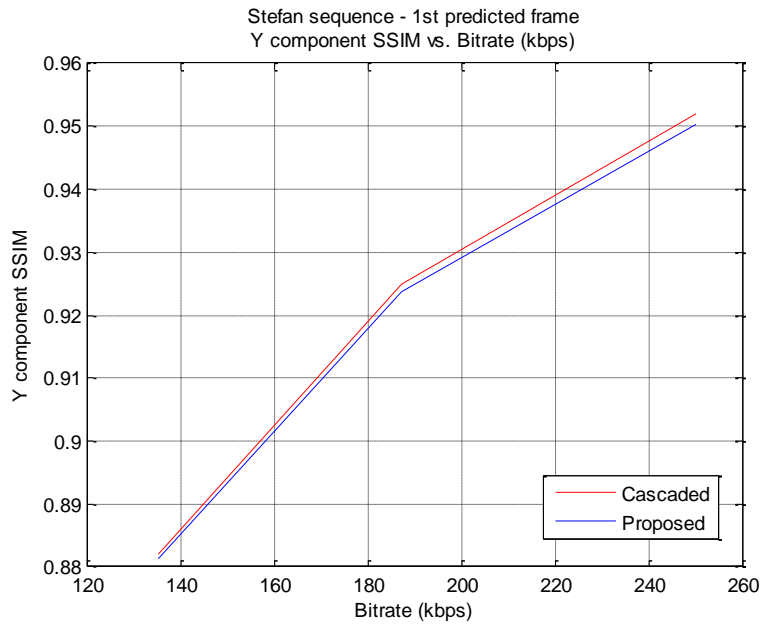
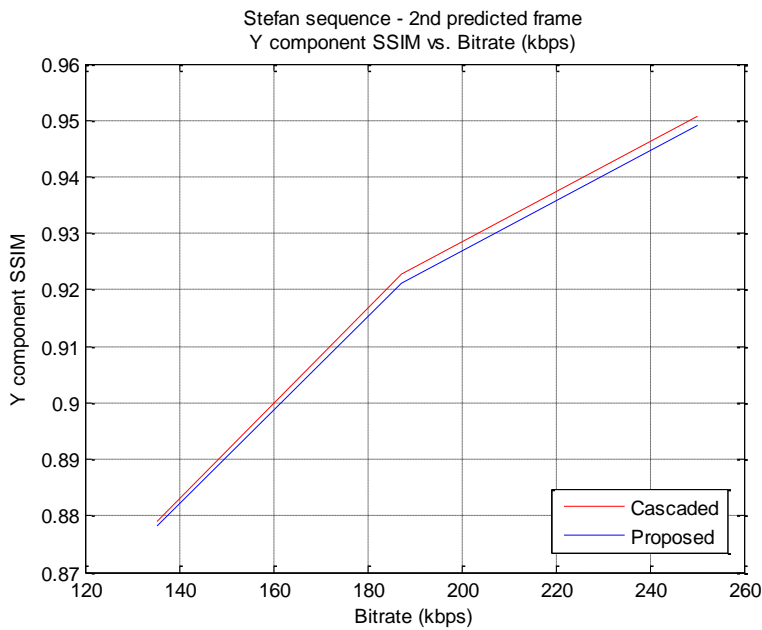


Figure 6.22 Stefan clip – PSNR (dB) vs bitrate (kbps) – Cascaded architecture and proposed technique



(a)



(b)

Figure 6.23 (a) Stefan sequence – Y component SSIM vs. Bitrate (kbps) – 1st predicted frame
(b) Stefan sequence – Y component SSIM vs. Bitrate (kbps) – 2nd predicted frame

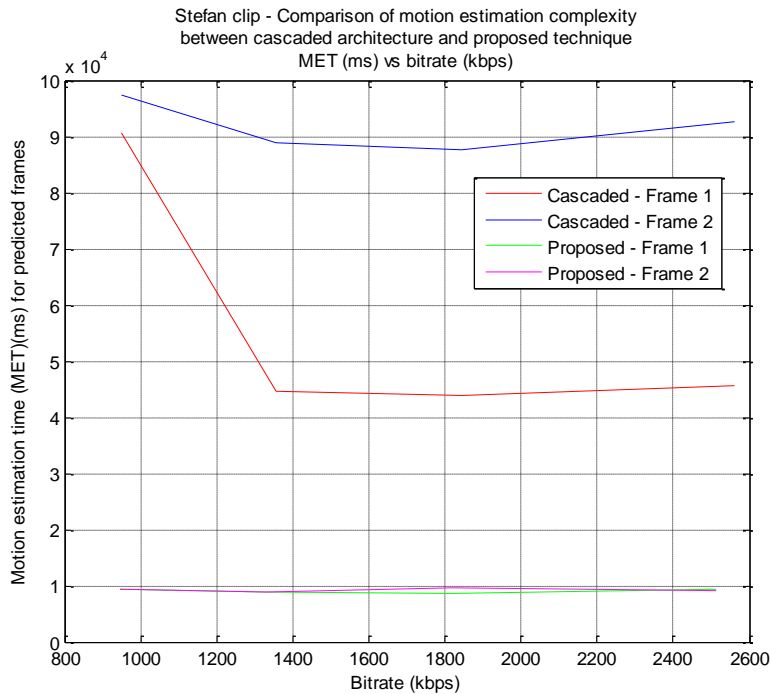


Figure 6.24 Stefan clip – Motion estimation time (MET) vs bitrate (kbps) – Cascaded architecture and proposed technique

6.3 Conclusions and future work

It can be observed that the motion vector information from the incoming VP6 bitstream can be used in re-encoding the VP6 bitstream to H.264 bitstream. The resulting loss of quality as a result, in comparison with the cascaded decoder and encoder model is very less. The quality reduction is mainly due to simplified motion estimation process of VP6 in comparison to H.264. However corresponding reduction in the motion estimation time (MET) is high. Hence the complexity in the re-encoding process is reduced significantly using the proposed technique.

The current technique only proposes the re-use of motion vectors available from the VP6 bitstream. It does not involve any motion vector (MV) refinement. MV refinement process is useful in getting more accurate motion vector values from the approximate motion vector values by making a simplified search in small windows around the approximate values. The authors in [48] and [51] describe different motion vector refinement techniques. The transcoded video

quality can be further improved by using MV refinement. Finding an appropriate MV refinement technique to supplement the proposed MV reuse can form the basis for future research. Different MV refinement techniques are described in [50] thru [52]. Section 5.2.5.4 also described some of these MV refinement techniques.

REFERENCES

- [1] R. Schafer and T. Sikora, "Digital video coding standards and their role in video communications," Proceedings of the IEEE Vol 83, pp. 907-923, Jan 1995.
- [2] E. Lallana and M. Uy, "The Information Age," UNDP-APDIP, 2003.
- [3] Open source article, "Digital Revolution," Wikipedia Foundation,
http://en.wikipedia.org/wiki/Digital_Revolution
- [4] K. Sayood, "Introduction to Data Compression," 3rd Edition, Morgan Kaufmann Publisher Inc., 2006.
- [5] ITU-T Recommendation H.264 – Advanced Video Coding for Generic Audio-Visual services.
- [6] A. Tamahankar and K. R. Rao, "An overview of H.264 / MPEG-4 part 10," Proc 4th EURASIP Conference focused on Video / Image Processing and Multimedia Communications, Zegreb, Croatia, pp 1-51, July 2003.
- [7] J. Emigh, "New Flash Player rises in the Web-Video Market," IEEE Computer, vol. 39, pp 14–16, 2006.
- [8] "Adobe Extends Web Video Leadership with H.264 Support", Adobe press release, August 21, 2007.
- [9] Joint Video Team (JVT), ITU-T website, <http://www.itu.int/ITU-T/studygroups/com16/jvt/>
- [10] S. 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.
- [11] T. Wiegand and G. J. Sullivan, "The H.264 video coding standard", IEEE Signal Processing Magazine, vol. 24, pp. 148-153, March 2007.

- [12] K. R. Rao and J. J. Hwang, "Techniques and standards for image, video and audio coding," Prentice Hall, 1996.
- [13] 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.
- [14] 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. 1 - 596, 11-14 Sept. 2005.
- [15] K. R. Rao and P. C. Yip, "The transform and data compression handbook", Boca Raton, FL: CRC press, 2001.
- [16] 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.
- [17] 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.
- [18] I. E. Richardson, "The H.264 Advanced Video Compression Standards," Second Edition, Wiley, Hoboken, NJ, May 2010.
- [19] I. E. Richardson, "H.264 / MPEG-4 Part 10: Intra Prediction," http://www.vcodex.com/files/h264_intrapred.pdf, April 2003.
- [20] I. E. Richardson, "H.264 / MPEG-4 Part 10: Inter Prediction," http://www.vcodex.com/files/h264_interpred.pdf, April 2003.
- [21] I. E. Richardson, "H.264 / MPEG-4 Part 10: 4x4 Transform and Quantization in H.264," <http://www.vcodex.com/h264transform4x4.html>, April 2009.
- [22] I. E. Richardson, "H.264 / MPEG-4 Part 10: Reconstruction Filter," http://www.vcodex.com/files/h264_loopfilter.pdf, April 2003.

- [23] J. Loomis and M. Wasson, "VC-1 Technical Overview,"
<http://www.microsoft.com/windows/windowsmedia/howto/articles/vc1techoverview.aspx>,
Microsoft Corporation, Oct. 2007.
- [24] "Real Video 10 – Technical Overview, version 1.0," Real Networks,
http://docs.real.com/docs/rn/rv10/RV10_Tech_Overview.pdf, 2003.
- [25] On2 Technologies, Inc., "White Paper – On2 VP6 for Flash 8 Video",
<http://www.On2.com>, Sept 12, 2005.
- [26] T. Uro, "The quest for a new video codec in Flash 8,"
<http://www.kaourantin.net/2005/08/quest-fornew-videocodec-in-flash-8.html>, August 13,
2005.
- [27] "VP6 bitstream and decoder specification," On2 Technologies Inc., Aug 2006
- [28] M. Vetterli and A. Ligtenberg "A Discrete Fourier-Cosine Transform Chip," IEEE Journal
on Selected Areas of Communications, Vol. SAC-4 No.1, pp 49-61, Jan. 1986.
- [29] I. Ahmad, et al, "Video Transcoding: An Overview of Various Techniques and Research
Issues", IEEE Trans. on Multimedia, vol 7, pp 793-804, October 2005
- [30] J. Xin, C. Lin and M. Sun, "Digital Video Transcoding", Proceedings of the IEEE, Vol.
93, pp 84-96, January 2005
- [31] A. Vetro, C. Christopoulos and H. Sun, "Video transcoding architectures and
techniques: an overview," IEEE Signal Processing Magazine, pp 18-29, March 2003.
- [32] Adobe press release, "Macromedia and Sorenson bring video to Macromedia Flash
content and applications,"
http://www.adobe.com/macromedia/proom/pr/2002/flash_mx_video.html, March 2002.
- [33] ITU-T Recommendation H.263 – Video coding for low bit-rate communication.
- [34] On2 Technologies Inc., "VP6 bit-stream overview – presentation."
- [35] On2 Technologies Inc., "On2 VP6 and H.264 for Adobe Flash Player,"
http://support.on2.com/files/h264_and_flash_faq.pdf, August 2007.

- [36] Tim Siglin, "On2 Technologies white paper: Flash video codec comparison," www.on2.com, July 2008.
- [37] H.264 and VP6 quality comparison for Hulu's implementation, <http://realworldvideocompression.com/2008/07/more-about-hulus-480p-quality/>, 2008
- [38] C. Holder and H. Kalva, "H.263 to VP6 transcoder," Visual communications and image processing 2008, Proc SPIE, vol 6822, pp. 68222B, Jan 2008.
- [39] H. Sun, X. Chen and T. Chiang, "Digital video transcoding for transmission and storage," CRC Press, 2005.
- [40] Open source article, "H.264/MPEG-4 AVC," Wikipedia foundation, http://en.wikipedia.org/wiki/H.264/MPEG-4_AVC
- [41] Open source article, "Blu-ray Disc," Wikipedia foundation, <http://en.wikipedia.org/wiki/Blu-ray>
- [42] B. Girod, "Overview: Video coding standards," Stanford University coursework, <http://www.stanford.edu/class/ee398b/handouts.htm>
- [43] G. Sullivan, "Overview of international video coding standards (preceding H.264/AVC)," ITU-T VICA workshop, Geneva, July 2005.
- [44] J. Xin, M. T. Sun, and K.-S. Kan, "Bit allocation for joint transcoding of multiple MPEG coded video streams," Proc. IEEE Int. Conf. Multimedia and Expo, pp. 8–11, 2001.
- [45] A. Vetro, H. Sung and Y. Wang, "Object-based transcoding for adaptable video content delivery," IEEE Trans. Circuits Syst. Video Technology, vol. 11, no. 3, pp. 387–401, Mar. 2001.
- [46] M.-T. Sun, T.-D. Wu and J.-N. Hwang, "Dynamic bit allocation in video combining for multipoint conferencing," IEEE Trans. Circuit Syst. II, vol. 45, no. 5, pp 644-648, May 1998.
- [47] O. Werner, "Requantization for transcoding of MPEG-2 intraframes," IEEE Trans. Image Processing, vol. 8, no. 2, pp. 179-191, Feb. 1999.

- [48] T. Shanabelah and M. Ghanbari, "Heterogeneous video transcoding to low spatial temporal resolutions and different encoding formats," *IEEE Trans. Multimedia*, vol. 2, no. 2, pp. 101-110, Jun. 2000.
- [49] K.-H. Tan and M. Ghanbari, "Layered image coding using the DCT pyramid," *IEEE Trans. Image Processing*, vol. 4, no. 4, pp. 512-516, Apr. 1995.
- [50] J.-N. Hwang and T.-D. Wu, "Motion vector re-estimation and dynamic frame-skipping for video transcoding," *Conf Rec. 32nd Asilomar Conf. Signals, Systems and Computer*, vol 2, pp 1606-1610, 1998.
- [51] J. Youn, M.-T. Sun and C.-W. Lin, "Motion vector refinement for high-performance transcoding," *IEEE Trans. Multimedia*, vol. 1, no. 1, pp 30-40, Mar 1999.
- [52] M.-J. Chen, M.-C. Chu and C.-W. Pan, "Efficient motion estimation algorithm for reduced frame-rate video transcoder," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 12, pp. 269-275, Apr 2002.

BIOGRAPHICAL INFORMATION

Jay R Padia is a graduate student of Electrical Engineering at University of Texas at Arlington. His research interests are image and video processing and visual computing. He has worked under the guidance of Dr. K. R. Rao at the Multimedia processing laboratory during the course of graduate studies. He has also been an intern at the Digital Home Group of Intel Corporation. Jay completed his undergraduate education at National Institute of Technology, Calicut in India after which he also worked for 2 years at Oracle Financial Services in Bangalore. He is joining the Visual computing group at Intel, Folsom, CA after graduation.