

**COMPARATIVE PERFORMANCE OF JM AND FFMPEG CODECS OF
H.264 AVC VIDEO COMPRESSION STANDARD**

A Thesis

Presented to the

Faculty of

San Diego State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in

Electrical Engineering

by

Hardik Sharma

Summer 2012

SAN DIEGO STATE UNIVERSITY

The Undersigned Faculty Committee Approves the

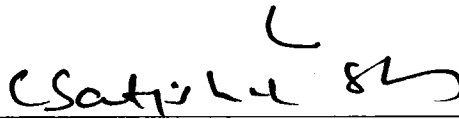
Thesis of Hardik Sharma:

Comparative Performance of JM and FFMPEG Codecs of H.264 AVC Video

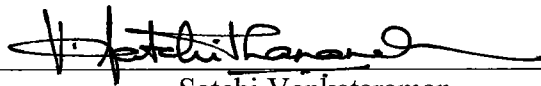
Compression Standard



Sunil Kumar, Chair
Department of Electrical Engineering



Satish Sharma
Department of Electrical Engineering



Satchi Venkataraman
Department of Mechanical Engineering

May 15, 2012

Approval Date

Copyright © 2012
by
Hardik Sharma
All Rights Reserved

DEDICATION

To my Family and God.

ABSTRACT OF THE THESIS

Comparative Performance of JM and FFMPEG Codecs of H.264
AVC Video Compression Standard

by

Hardik Sharma

Master of Science in Electrical Engineering

San Diego State University, 2012

With the rapid growth of wireless multimedia services, the demand for video transmission over wireless channels has been increasingly exponentially. This has led to the design of error resiliency schemes for obtaining better codecs. The perceptual video quality is influenced not only by the compression artifacts, but also by the channel errors. Hence, a video codec needs to accommodate contradictory requirements: coding efficiency and robustness to data loss, along with other limitations such as memory, bandwidth, real time coding and complexity.

In this thesis, we have studied the comparative performance of two prominent open source H.264 AVC video codecs - Joint Model (JM) and Fast Forward Motion Picture Expert Group (FFMPEG). We have studied various features of these codecs, such as macroblock tree rate control, multi-pass encoding, and psychovisual optimization model (it includes the parameters like adaptive quantization, psy-RD and psy-trellis). Observations based on the PSNR values and rate-control have been used to analyze the objective video quality for error-free and corrupted bitstreams generated by both codecs. In addition, we have analyzed the creation of different priorities based on cumulative mean squared error (CMSE) values for dividing the I, P and B video frames into four priorities. These studies helped us in understanding which codec can be more robust against channel errors or losses and how we can improve their performance.

TABLE OF CONTENTS

	PAGE
ABSTRACT.....	v
LIST OF TABLES	viii
LIST OF FIGURES	ix
LIST OF ACRONYMS	xi
ACKNOWLEDGEMENTS.....	xii
 CHAPTER	
1 INTRODUCTION	1
2 OVERVIEW OF H.264/AVC VIDEO CODING STANDARD.....	3
2.1 Network Abstraction Layer (NAL).....	3
2.1.1 NAL Units.....	4
2.1.2 NALU - Byte Stream Format.....	4
2.1.3 NALU – Packet Format	5
2.1.4 Parameter Set	5
2.2 Video Coding Layer (VCL)	5
2.3 Key Features of H264	7
3 OVERVIEW OF FFMPEG AND x264.....	10
3.1 Main Features – x264.....	10
3.2 FFMPEG Features and Parameter Set	22
4 DESCRIPTION OF SYSTEM SET UP	26
4.1 Introduction.....	26
4.2 Simulation Environment for JM	26
4.2.1 Encoder Parameter	26
4.2.2 Decoder Parameter	27
4.3 Simulation Environment for FFMPEG.....	27
4.3.1 Encoder Parameters	27
4.3.2 Decoder Parameter	28
5 EXPERIMENTAL EVALUATION OF FFMPEG AND JM CODEC	29

5.1 Introduction.....	29
5.2 Experiment and Comparison Without Losses.....	29
5.3 Experiments and Comparison with Losses	31
5.3.1 Enhancements to FFMPEG.....	32
5.3.2 Results for Weighted Prediction and Trellis Parameter in FFMPEG	34
5.3.3 Results for Different Loss Patterns in FFMPEG vs. JM.....	37
5.4 Priority Scheme Implemented on JM and FFMPEG Slices.....	38
6 CONCLUSION AND FUTURE STUDY	46
REFERENCES	47
APPENDIX	
A JM (H.264)	50
B FFMPEG (H.264)	53
C PROGRAM MODULES	57

LIST OF TABLES

	PAGE
Table 5.1. Results of BUS Sequence with Single Pass and Without Errors (FFMPEG).....	30
Table 5.2. Results of BUS Sequence with Two Pass and Without Errors (FFMPEG).....	31
Table 5.3. Results of BUS Sequence Without Errors (JM)	32
Table 5.4. Results of News Sequence with Single Pass and Without Errors (FFMPEG)	33
Table 5.5. Results of News Sequence with Two Pass and Without Errors (FFMPEG)	34
Table 5.6. Results of News Sequence Without Errors (JM)	35
Table 5.7. Results of Bus_CIF Sequence with 2% and 5% Errors in FFMPEG and JM	37
Table 5.8. Results of Foreman_CIF Sequence with 2% and 5% Errors in FFMPEG and JM.....	38
Table 5.9. Bus Showing Percentage I, P and B Slices in Different Priorities for Different Bit Rates in FFMPEG and JM	44
Table 5.10. GOP 20 of Foreman Sequence Showing Percentage of I, P and B Slices in Different Priorities for Different Bit Rates in FFMPEG and JM.....	45

LIST OF FIGURES

	PAGE
Figure 2.1. Layered structure architecture of H.264 codec.....	3
Figure 2.2. NAL header structure.	4
Figure 2.3. Motion estimation of current frame from multiple previous reference frames.....	7
Figure 2.4. Macroblock segmentation for motion compensation.	8
Figure 3.1. Diamond motion estimation search pattern.	19
Figure 3.2. Illustrates the hexagon motion estimation search pattern.....	19
Figure 3.3. Replace exagon to uneven multi-hexagon.....	20
Figure 3.4. Illustrates the exhaustive search pattern.	21
Figure 5.1. Illustration of how to introduce slice errors in FFMPEG coder.	36
Figure 5.2. Average PSNR comparison of JM and FFMPEG by enabling weighted prediction and disabling trellis parameter in FFMPEG.	36
Figure 5.3. PSNR values of frames for JM vs FFMPEG with 2% slice loss for 256 kbps Bus sequence.	38
Figure 5.4. PSNR values of frames for JM vs FFMPEG with 5% slice loss for 256 kbps Bus sequence.	39
Figure 5.5. PSNR values of frames for JM vs FFMPEG with 2% slice loss for 512 kbps Bus sequence.	39
Figure 5.6. PSNR values of frames for JM vs FFMPEG with 5% slice loss for 512 kbps Bus sequence.	40
Figure 5.7. PSNR values of frames for JM vs FFMPEG with 2% slice loss for 1024 kbps Bus sequence.	40
Figure 5.8. PSNR values of frames for JM vs FFMPEG with 5% slice loss for 1024 kbps Bus sequence.	41
Figure 5.9. PSNR values of frames for JM vs FFMPEG with 2% slice loss for 512 kbps Foreman sequence.	41
Figure 5.10. PSNR values of frames for JM vs FFMPEG with 5% slice loss for 512 kbps Foreman sequence.	42
Figure 5.11. PSNR values of frames for JM vs FFMPEG with 2% slice loss for 1024 kbps Foreman sequence.	42

Figure 5.12. PSNR values of frames for JM vs FFMPEG with 5% slice loss for 1024 kbps Foreman sequence.	43
Figure 5.13. Description of the CMSE calculation process for FFMPEG.....	43

LIST OF ACRONYMS

CAVLC:	Context Adaptive Variable Length Coding
CABAC:	Context Adaptive Binary Arithmetic Coding
IDR:	Instantaneous Decoder Refresh
NAL:	Network Abstraction Layer
NALU:	Network Abstraction Layer Unit
PSNR:	Peak Signal to Noise Ratio
VQM:	Video Quality Metric
GOP:	Group Of Picture
FMO:	Flexible Macroblock Ordering
DP:	Data Partition
RDO:	Rate Distortion Optimization

ACKNOWLEDGEMENTS

Firstly, I would like to take the opportunity to sincerely thank my thesis guide, professor Dr. Sunil Kumar for his continuous support and mentorship during my thesis research work. Right from the very first day, he has encouraged and guided me towards exploring and polishing my skills as a researcher. His support and advice helped me learn and understand different concepts of multimedia design. He has worked with me very patiently and supported me through all means possible.

I also want to express my appreciation to the expert's community on the FFMPEG mailing list for valuable information and time given to clarify my countless queries.

I would also like to extend my gratitude to my committee members, Dr. Satish Sharma and Dr. Venkat Satchiraman, for their time and generous agreement to be a part of my thesis committee.

Furthermore, I would like to thank all my colleagues from the Multimedia and Wireless Networks lab of SDSU who have helped me immensely in the past few years.

I would also like to express my deepest appreciation to my parents who provided me with silent but crucial help throughout the completion of this project.

Finally, I express my sense of sincere gratitude to Almighty God for bestowing me with knowledge and skills that helped me complete my thesis work.

CHAPTER 1

INTRODUCTION

H.264/AVC (Advanced Video Coding) is the latest video standard jointly developed by the ITU-T Video Coding Experts Group (VCEG) and the ISO/IEC Moving Picture Experts Group (MPEG). This video standard can achieve up to 50% improvement in bit-rate efficiency compared to the previous video coding standards, such as MPEG-4. Because of its high compression, hybrid video coding schemes and network friendliness, it has been adopted by many application standards such as HD DVD, DVB-H, HD-DTV, etc. [1]

The demand for video transmission over network leads to complications like instability in network behavior, which is a problem that needs to be addressed through the error protection methods for encoded video data over network. In real time applications, this need of data protection very high, as the re-transmission of lost or corrupted video packets may not be possible.

Various error protection schemes - including equal error protection (EEP) as well as unequal error protection schemes (UEP) schemes - help immensely against all kinds of errors or video packet drops that can occur due to the dynamic nature of network. These schemes help in minimizing the errors in more important video data and thus getting better quality even in noisy network environments. Depending on the need of the environment, the error protection schemes use different features for a given video sequence.

Packet size is very important in designing error protection schemes [2]. The smaller the packet size, the lesser its chances are to become affected by errors [3]. However, a smaller packet size reduces the compression efficiency of the encoder, which is not desirable. Hence, designing an error protection scheme for a particular network involves a trade-off between different competing requirements. The features that are considered useful for these protection schemes include slice grouping, data partitioning, frame type, frame location, and packet size. The idea is to use these features in deciding which data is more important when it comes to achieving better quality in the case of a noisy network environment. Using these

features, we can form packets of data, provide the importance of these packets in a priority list, and give protection to packets proportional to their importance.

In this thesis, we study the features of FFMPEG and their importance for H.264 coded bitstreams. We compare the results of FFMPEG codec with JM codec based on various outputs such as PSNR, VQM, decoding-encoding time and the capability to achieve target bitrate. The thesis is organized in the following chapters.

We provide the overview of H.264/AVC video coding standard in Chapter 2. It explains the working of network abstraction layer (NAL) and video coding layer in H.264. This chapter explains the NAL unit (NALU) including picture parameter set and sequence parameter set. It also explains some key features of H.264 such as the in-loop deblocking filter, profiles, etc. Chapter 3 explains the FFMPEG and x264 features such as the rate control - which includes constant quantizer, constant quality, single or multipass ABR and optional VBV - Scene cut detection, Psy optimizations for detail retention, which includes adaptive quantization, psy-RD and psy-trellis, Multi-pass encoding, etc.

All the encoding and decoding parameter settings for our experiments are explained in Chapter 4, including the simulation environment for JM and FFMPEG both. In Chapter 5, we discuss all the experiments, with and without channel-induced losses, and their results for the comparison of JM and FFMPEG codecs. The conclusion of our work is given in Chapter 6.

CHAPTER 2

OVERVIEW OF H.264/AVC VIDEO CODING STANDARD

H.264 is the leading and most advanced video codec currently available as it provides the most efficient coding method that gives the best quality while using considerably less bits. Figure 2.1 depicts the layered structure of H.264. Its layered structure provides two major parts, of which the first is NAL (Network Abstraction Layer) and the other is VCL (Video Coding Layer).

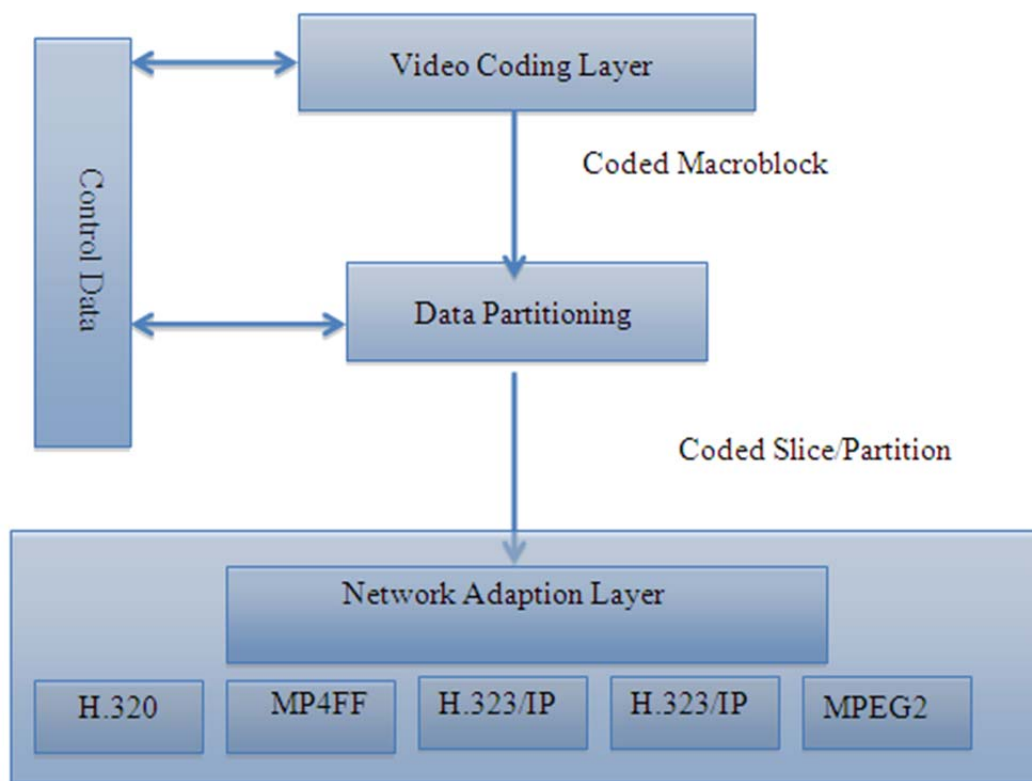


Figure 2.1. Layered structure architecture of H.264 codec.

2.1 NETWORK ABSTRACTION LAYER (NAL)

The NAL [4] in the H.264 video coding standard provides outside networks with the flexibility in using VCL. NAL takes care of the mapping of VCL data on the network while

providing maximum benefits for encoded VCL data by preparing the data for the worst conditions on the network. It also maps data for many transport layers such as RTP/IP, Internet services, file formats, etc.

2.1.1 NAL Units

The coded video data from H.264 encoder is arranged in NAL units (NALU), which contain integer bytes. The first byte in each NAL unit contains a header and the start code prefix is 0x000001; the remaining data is payload. The header byte in NALU contains error flag, disposable NALU code, and NALU type, including one forbidden bit, two nal_ref_idc bits (which tell the decoder whether or not that particular unit should be used for reference), and five bits as nal_unit_type to indicate the type of NALU. Figure 2.2 illustrates the same. The header provides the connection between the NAL and VCL. NAL unit structure provides the flexibility to the generic structure where it can be used in the formats as packet oriented and bitstream-oriented transport systems.

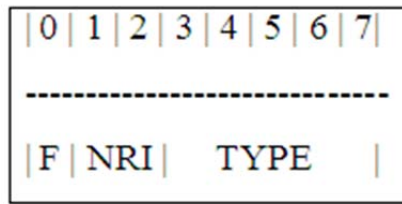


Figure 2.2. NAL header structure.

2.1.2 NALU - Byte Stream Format

Considering the requirement of some systems (e.g., MPEG-2 TS, H.320 etc.), which support byte stream data as inputs, H.264 developers included within it the capability to generate byte stream encoded data. As per the requirement from systems, H.264 generates the entire data in NAL bytes, including the boundary codes or suffix codes for identifying the boundaries of different units in that whole byte stream. This start code is totally unique, which makes sure that the start of a new NALU is identifiable. It also provides one extra byte per picture for the decoder to align the whole byte stream data by looking from outside. The encoder can add more bytes, or additional information in the form of extra bytes, to the decoder for fast byte data alignment; it can also be used to expand the data sent to decoder in byte stream format.

2.1.3 NALU – Packet Format

Some other systems like IP/RTP use packet formats. Therefore, H.264 also supports the encoded data packets, which are further framed in formats that are compatible with given system transfer protocols. Due to the use of packet formats for NAL units, unique boundary codes are not required.

2.1.4 Parameter Set

Parameter sets give information about the encoded video data to the decoder, which helps in increasing the robustness of the data as well. There are two types of parameter sets available in H.264: (i) Sequence Parameter set (SPS), and (ii) Picture Parameter set (PPS).

A sequence parameter set (SPS) provides the header information for all NAL units of a coded sequence; it gives information to the decoder regarding consecutive coded frames/pictures. A picture parameter set (PPS) provides information about each and every frame/picture during the decoding of a given video sequence.

SPS and PPS units can be sent to the decoder way ahead of the payload or coded data. Every NAL unit contains one identifier by which decoder can refer to the relevant PPS; every picture parameter set also contains one identifier which refers to its relevant SPS. By this means, a small amount of identifier helps in avoiding the repeat of a large amount of PPS data for reference in every NAL unit. Parameter set information provides robustness to the coded data to counter data loss due to errors during transmission.

2.2 VIDEO CODING LAYER (VCL)

Video coding layer represents the encoded data, which is the value of the samples in video pictures. This layer in the encoder carries out the encoding task for a given video data. Video coding layer is behind the block-based hybrid encoding approach of H.264 in which every picture is represented by block shaped units including luma and chroma samples known *macroblocks* (most basic unit in H.264). A luma component (Y) in a macroblock represents the brightness information, which is most important as per the psychovisual model. The chroma components (Cb, Cr) represent the color information. The major parts in video coding layer are as follows:

- Each picture in VCL unit is subdivided into rectangular boxes known as macroblock, which can vary from 16x16 to 4x4 in terms of size. These macroblocks can be intra-coded or inter-coded.
- The sequence of encoded macroblocks gets grouped to generate slices. A picture or frame is a combination of one or more slices. The information about slices is provided by parameter sets. A slice is an independent entity in H.264 codec, which means that it can be transmitted separately as well as decoded on its own.
- FMO is a feature for more robustness of H.264 encoded data that decides macroblocks are grouped to generate slices, and hence, frame.
- Other than FMO, H.264 uses various coding types for data; those types are I-type, P-type, and B-type. I-type slice coding provides the capability of intra-coding for all the macroblocks in a given slice. In P-type coding, macroblocks can either be inter-coded or intra-coded with the constraint of at most one motion compensated prediction signal per prediction block. This feature balances compression and robustness in P-type slices and macroblocks. B-type coded slices or macroblocks can also use the inter-coding method and can have two motion compensated prediction signals.
- Intra-prediction takes place by predicting the samples of macroblocks with the available information about already transmitted macroblocks of the same frame.
- Inter-motion compensated prediction takes place for macroblocks from already transmitted reference frames. To get more accuracy for this prediction mode, macroblocks can go up to sub-macroblock level.
- Transform coding is used in the H.264 video standard to explore the spatial redundancies present in a frame to get maximum possible compression by allotting minimum bits. H.264 implements integer transform with same properties as the 4x4 DCT. Since it is defined as exact integer operation, its inverse-transform mismatches can be avoided.
- Generally transform are of 4x4 size but in some special cases, it decreases to 2x2 (i.e., for chrominance).
- The block sizes of 4x4 transform size help in the adaptation of prediction error coding to the boundaries of fast moving objects, and hence, less noise around edges.
- Matching the block size of motion compensation provides more flexibility.
- Also, the smaller the transform, the lesser the computation and processing word length.
- After quantization, the transform coefficients for a block get scanned and then transmitted using CABAC and CAVLC entropy coding.
- CAVLC – It is a very efficient entropy coding method for the transmitting of quantized transform coefficients. This entropy method is not very complex and isn't as much as a decoding process as context-adaptive binary arithmetic coding (CABAC) is. This is available as the default entropy coding method in all the profiles for the coding of DC and AC coefficients.

- **CABAC** – This method is more complex than CAVLC and needs much more processing at the decoder side. But at the same time, it provides better efficiency. This entropy coding method is only available in the main profile (MP) and high profile (HP) of the H.264 coding standard.
- **In-loop deblocking filter** – A big disadvantage of the block based coding method (blocking artifacts) in the H.264 codec is removed (or mostly reduced) by including the In-loop deblocking filter [5]. It works very efficiently on edges to remove the relatively large absolute difference between samples. By using this filter, the blockiness gets reduced while sharpness remains same.

2.3 KEY FEATURES OF H264

The increasing demand for video over wireless networks has also increased the other needs for a video coding standard, such as better compression and flexibility. The main challenge for video transmission over a network has been the limited channel capacity and the unexpected channel behavior [6]. To overcome these problems, the video encoders exploit the spatial and temporal redundancies present between the frames. H.264 AVC is one such video coding standard jointly developed by ISO and ITU. H.264 AVC has many key features as discussed below, which achieve 50% higher compression ratio, and make it network friendly and more robust against channels errors and packet losses [4, 7, 8]:

- **Multiple reference frames** – In H.264, the number of reference frames can go up to 16, much larger than in previous codecs which only had 1 reference frame. This feature greatly helps H.264 in improving bitrate and boosting the quality of data encoding. Figure 2.3 illustrates the motion compensation for a macroblock in the current frame from multiple previous reference frames. [9]

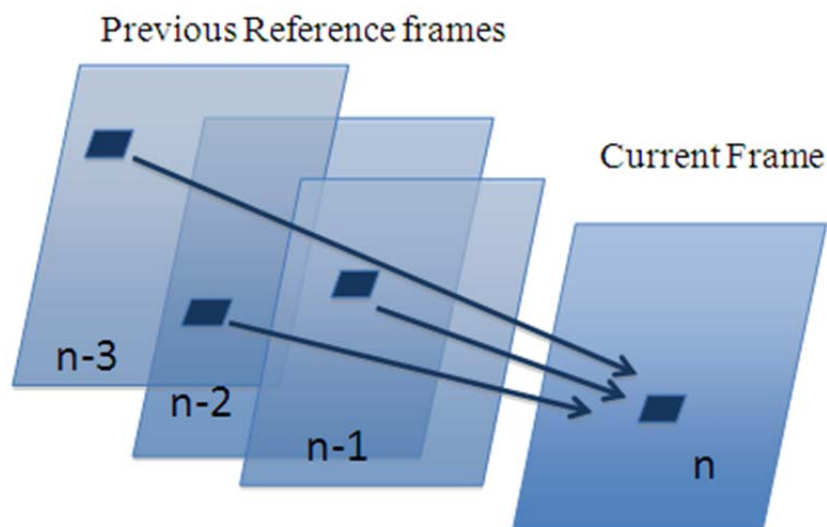


Figure 2.3. Motion estimation of current frame from multiple previous reference frames.

- **Variable block size motion compensation** – H.264 provides more flexibility to the motion compensation block sizes. The block sizes for macroblocks can vary from 16x16 to 4x4, including all the combinations (i.e. 16x8, 8x16, 8x4, 4x8, 8x8, 16x4, etc.) as in Figure 2.4.

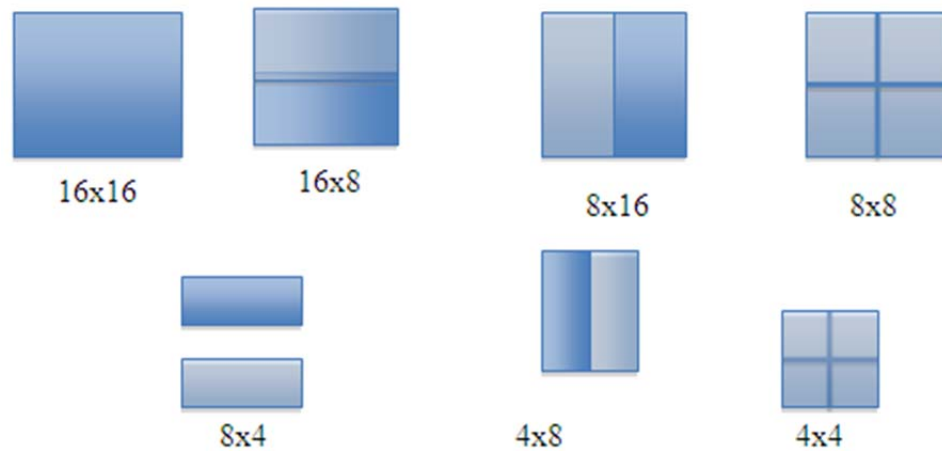


Figure 2.4. Macroblock segmentation for motion compensation.

- Unlike previous coding standards, the referencing order and display order are not dependent on each other, which give much flexibility to the encoder. Removal of this coupling also removed the extra delay previously associated with bi-predictive coding.
- **Weighted prediction** – This new feature of H.264 improves the coding efficiency significantly. This feature allows for weighted motion compensated prediction signal as well as offset by amounts that are specified by the encoder.
- H.264 includes two types of entropy coding for maximum efficiency - CAVLC (context-adaptive variable length coding) and CABAC (context-adaptive binary arithmetic coding). CABAC gives the best compression but it is more complex than CAVLC.
- The picture parameter set (PPS) and the sequence parameter set (SPS) give information about the picture and the whole video sequence, respectively. The information obtained through these parameters gives more robustness to the coded data because with the loss of some data in sequence, the decoder can still get information to decode the rest and use concealment, depending on the information obtained.
- H.264 includes many error resiliency features like FMO (flexible macroblock ordering), which provides macroblock ordering in various patterns to provide robustness to the encoded data by managing a spatial relationship between the regions, flexible slice size, which gives the flexibility to make packet size and include a particular type of slice in it, ASO (Arbitrary slice ordering) where every slice is independent of another gives flexibility in sending and receiving the slices of a picture in any order, Data partitioning, and redundant picture.

- **SP/SI synchronization pictures** – This new and innovative feature of H.264 gives the perfect switching or synchronizing points from an ongoing decoding process of a given video sequence.
- The following profiles in H.264/AVC define the set of different enabled features:
 1. **Baseline Profile (BP)** – This profile does not support B frames and CABAC. To support error resiliency and robustness, this profile supports FMO, ASO, and redundant slices. This is mainly used for video conferencing and mobile applications.
 2. **Extended Profile (XP)** – This profile supports FMO, ASO, redundant slices, Data Partitioning, SI and SP switching slices, and B slices. This is used extensively for streaming purposes. This profile also provides higher compression capabilities.
 3. **Main Profile (MP)** – This profile supports the CABAC and CAVLC entropy coding but does not support FMO, ASO, redundant slices, data partitioning, and SI and SP slices. This is used for standard definition digital TV broadcasts.
 4. **High Profile (HP)** – This profile in H.264 is very similar to Main Profile (MP) but has a few additional features, which are QP (quantization parameter) for separate chroma components. In this profile, the transform adaptivity is also enabled for 8x8 vs. 4x4. This is used for broadcast and disc storage applications and specifically for high definition television applications.

CHAPTER 3

OVERVIEW OF FFMPEG AND x264

FFMPEG [10] is an open source transcoder that contains libraries and programs for handling multimedia data. FFMPEG stands for ‘fast forward MPEG’. It supports many codecs and formats of video, audio and images, including external libraries to support some formats. The most notable parts of FFMPEG are libavcodec, an audio/video codec library used by several other projects, libavformat, an audio/video container mux and demux library, and the FFMPEG command line program for transcoding multimedia files [11]. FFMPEG is based on Linux but its recent versions can also be supported in Windows.

For H.264 video encoding, FFMPEG uses the features available in the x264 codec. The H.264 decoder in FFMPEG has its own library. x264 is a free software library for encoding video streams in H.264/MPEG formats. x264 is one of the best encoders available for real time encoding.

3.1 MAIN FEATURES – x264

x264 has incorporated many really good features [12] for H.264 video standard which provide better visual quality than other available H.264 encoders. Some important features are as follows:

- Adaptive B frame placement
- Usage of B frames for referencing or arbitrary ordering
- CAVLC/CABAC encoding
- 8x8 and 4x4 adaptive spatial transform
- Custom quantization matrices
- Intra: all macroblock types (16x16, 8x8, 4x4, and PCM with all predictions)
- Inter P: all partitions (from 16x16 down to 4x4)
- Inter B: partitions from 16x16 down to 8x8 (including skip/direct)
- Interlacing (MBAFF)
- Multiple reference frames

- Rate control: constant quantizer, constant quality, single or multipass ABR, optional VBV
- Scene cut detection
- Spatial and temporal direct mode in B-frames, adaptive mode selection
- Parallel encoding on multiple CPUs
- Predictive lossless mode
- Psy optimizations for detail retention (adaptive quantization, psy-RD, psy-trellis)
- Zones for arbitrarily adjusting bitrate distribution
- Multi-pass encoding

Explanation for features and parameter set is as follows [13]:

Profiles- Profiles give the required settings to an output stream. Once we set any profile for output stream it overrides all previous settings of the user. Profile settings should be used according to the decoding device. Profiles supported by x264 are Baseline, Main and High.

Example for command line:

--profile main

Preset- This parameter in x264 takes over compression efficiency. It trades off between compression and encoding speed. The slower the speed, the better the compression efficiency. Presets supported by x264 are ultrafast, superfast, very fast, faster, fast, medium, slow, slower, very slow.

Example for command line:

--preset fast

Keyint- It defines GOP length by providing the difference between two IDR/I frames for the whole video sequence. IDR frames work as delimiters because no frame can take reference from the frames previous to an IDR frame. It can be kept in such a way that the scene-cut detection can decide where to put an IDR frame. It can help compress videos containing very slow movement.

Example for command line:

--keyint 250 //GOP length of 250 frames

Min-keyint- This is an important parameter for scene-cut detection along with keyint. By default, it is calculated as (keyint/10). It controls the minimum availability possible of an

IDR frame in a video sequence. The maximum value allowed for min-keyint is $((\text{keyint}/2)+1)$.

Example for command line:

--min-keyint 25

Scene cut detection- This feature of x264 allows the encoder to put key IDR/I frames according to the calculation of a metric for scene cut detection. This metric in x264 calculates how different the current frame is from the previous frame. If it is more than the percentage mentioned with this parameter, it replaces the frame with an IDR or I key frame. It replaces I or IDR in such a manner that if it is less than min-keyint since the last IDR, the I frame is placed; otherwise, an IDR frame is placed.

Example for command line:

--scenecut 40

Adaptive B frame placement- x264 uses an algorithm to decide for adaptive B frame placement in the encoding sequence. This provides authority to select between P frame or B frame. This feature affects the encoding speed by choosing a higher number of B frames or P frames, depending on the setting of --b-adapt parameter. It has three modes:

First mode is represented by the b-adapt value '0' which disables the adaptive B frame algorithm and chooses B frame always.

Second mode is represented by the b-adapt value '1' which provides fast algorithm and wants to use the 16 B frame period. It is also the default parameter that x264 uses for adaptive B frame placement.

Third mode uses the b-adapt value of '2'; it is a slow but optimal algorithm to decide between the P and B frames and make a selection. In multipass encoding, this feature works only in the first pass where the encoder is selecting the types.

Example for command line:

--b-adapt 1 (default)

B frame referencing- In x264, we can decide whether or not the encoder can use the B frame as a reference for other frames. The parameter that enables this feature is b-pyramid. By using B frame as reference, the quantizer can come almost halfway from P to B. There can be two types of B frames available after encoding. B frame can be represented by the symbol 'B' or by 'b' in the display order. The difference between them lies in the fact that

‘B’ can be used as reference while ‘b’ cannot. Also, to use this parameter, either there should be at least 2 b-frames or there should be 2 B frames already available before starting to use B frame as reference.

Modes to use b-pyramid parameter in x264 encoding are as follows:

Mode ‘none’- This mode does not allow b frames to be used as reference. So, all the B frames are going to be represented by ‘b’ in display order.

Mode ‘normal’- This mode allows for many B frames to be used as reference in GOP. This is a default setup during encoding.

Mode ‘strict’- This mode allows only one B frame to be used as reference in the GOP.

Example for command line:

--b-pyramid normal

Slices- This parameter allows for the input of the number of slices needed in every frame. It is overridden by the slice-max-size or slice-max-mbs, if already defined during encoding. **Slice-max-size** allows the user to define the maximum size of every slice. **Slice-max-mbs** allows the user to define the maximum number of macroblocks in every slice. The default value for all of them is 0.

Example for command line:

--slices 5 // 5 slices per frame

--slice-max-size 150 // slice size in a frame should not exceed 150 bytes

--slice-max-mbs 200 //No slice should contain more than 200 macroblocks.

Interlaced- x264 supports interlaced video encoding format by some parameters, tff and bff. ‘tff’ specify the top field first and ‘bff’ specify the bottom field first. Users can disable it by enabling the parameter- ‘no-interlaced’.

Rate control- Rate control in video encoding is used to get the best and consistent quality possible with controlled bitrate and QP (quantization parameter) values or user-defined values of these parameters. The implementation of rate control in H264 video standard can be on a whole GOP level, frame level, slice level, or macroblock level. On different levels, the rate control algorithm determines QP values for transform coefficients. The x264 rate control algorithm depends on the implementation of libavcodec, which is mostly empirical [14]. Rate control in x264 can be done in five different ways which are

dependent on the multipass (i.e., generally 2 pass encoding) and single pass encoding modes as described below:

Multipass mode (two-pass)- In this approach x264 first runs a quick encoding for the target video sequence, and then uses the data from this run for a second pass encoding in order to achieve the target bitrate by distributing the bits appropriately.

- Starting the second pass decides the relative number of bits that are going to be allocated between different frames, independent of the total number of bits for the whole encoding. The formula to empirically select the value is $\text{complexity}^{0.6}$, where complexity is the approximate bit size of a frame at constant QP. Above approach gives us the value of QP for P frames and how it can be used to decide QP values for nearby I and B frames.
- Then scale down or up the QP value from the above step to achieve the total target bits.
- While encoding a frame, the QP values for future frames are updated to take care of any deviations from target size. If in the second pass, the real size consistently deviates too much from the target size, the target size of all upcoming frames is multiplied by the ratio of the predicted/target file size; the real file size or the qscale of all future frames can be multiplied by the reciprocal of error.

The above approach is known as long-term compensation. The short-term compensation can be helpful at the early stages of this deviation or at the very end (where the cost to use long-term compensation cannot be justified); it uses the compensation factor $C^{(\text{real file size}-\text{target file size})}$, where C is a user defined constant value. Above approach can go beyond 2-pass encoding.

Single pass average bitrate mode (ABR) - This is a one-pass mode where the aim is to get a bit rate as close as possible to the target bitrate, and hence, file size. There is no benefit of knowing the data for future frames because there is only one-pass available to perform the entire encoding process.

- The first step is the same as in 2-pass but here, we do not have the complexity of future frames. To compensate for that, it runs a fast motion estimation algorithm on half resolution of each frame and considers the sum of absolute Hadamard transform difference (SATD) residuals to check on complexity. As there is no information regarding the complexity of a future group of pictures, the QP value of I frame depends on the past.
- There is no prediction of complexities for future frames, so it should scale based on the values from the past alone. The scaling factor is chosen to one, which had given the desired values for past frames.

- Both long-term and short-term compensation is the same in this mode as in the 2-pass mode. By tuning the strength of compensation, it is possible to obtain quality ranging from close to 2-pass (but with file size error of $\pm 10\%$) to lower quality with strict file size.

Example for single pass command line:

--bitrate 512k.

Single pass constant bitrate (VBV compliant) - This single pass mode is to achieve constant bitrate and is especially designed for real time streaming.

- It calculates the complexity estimation of frames in the same manner that is used for computing bit size as the ABR mode above.
- In this mode, the decision for scaling factor takes place based on the past values from the frames in the buffer instead of all the past frames. This value also depends on the buffer size.
- The overflow compensation works in a similar manner as ABR and the above mode; the only difference is that it runs for every row of macroblocks in the frame instead of for whole frames like in previous modes.

Example for command line:

--vbr-maxrate, --vbr-bufsize, --vbr-init

Single pass constant rate factor (CRF) - This single pass mode works with the user-defined value for constant rate factor/quality instead of bitrate. The scaling factor is constant, based on the --crf argument which defines the quality requirement from the user. There is no overflow compensation available in this mode.

Example for command line:

--crf 2

Single pass constant quantizer (CQP) - In this mode, the QP value depends on whether the current frame type is I, B or P frame. This mode can only be used when the rate control option is disabled.

Example for command line:

--qp 28

Rate tolerance- The --ratetol parameter gives the encoder flexibility in the 1st encode of multipass encoding mode to miss the target bitrate by a user-defined percentage to obtain better quality. The value of this parameter can vary from 0.01 to any number, where 1 means 1% deviation from target rate. This parameter helps in the situation where at a particular point in a video sequence, some high motion scenes are coming and the one-pass encoder has

no idea that a higher number of bits are going to be needed near the end. However, it also affects the file size. So, there is a tradeoff between quality and size.

Example for command line:

--ratetol 1.0

Psycho-visual enhancement features- Psycho-visual enhancement is a very important and helpful feature that enhances the visual or subjective quality of the encoded frames. This feature depends on the following three major parts:

- Adaptive Quantization (AQ) - The purpose of adaptive quantization is to avoid the blocking in flat areas, which is caused by the variation of different macroblock partition values from 16x16 to 4x4, depending on the complexity level of frame. More importantly, adaptive quantization avoids blurring in relatively flat textured areas like football fields. If AQ is disabled by the user during encoding in x264, it ends up giving less bits to less detailed sections of the frame. Users can define this parameter as follows:

--aq-mode 1 or 2 or 0 (1 is by default)

Here, Mode '0' disables AQ, Mode '1' allows the parameter to re-allot the bits to whole sequence or any particular frame, and Mode '2' allows AQ to gain strength per frame and decide based on that.

- Macroblock tree rate control (MB tree) - MB tree rate controls the quality by tracking how often parts of the frame are used for predicting the future frames. In this feature, there is a need of future frame prediction, which is where it requires multi-pass encoding where it can get future values for the prediction from the first pass and then apply the algorithm on the next pass to do rate control. It tracks the propagation of information from future blocks to past blocks across the motion vectors. It can be described as localizing quantization compression curve (--qcomp) to act on individual blocks instead of on the whole frame. Thus, instead of lowering quality in the entire high complexity frames, it will only lower quality on the complex part of the frame. This feature can help phenomenally on very low bit rates (for example, 67kbps animated clip).

User can disable mb tree rate control as follows-

--no-mbtree 1

- Trellis quantization- Trellis quantization is an algorithm that can improve data compression in DCT (discrete cosine transform) based encoding methods. It is used to optimize residual DCT coefficients after motion estimation in lossy video compression. Trellis quantization reduces the size of some DCT while recovering others to take their place. It increases the quality because the coefficients chosen by trellis have the lowest rate distortion ratio.

User can define the value for trellis as follows:

--trellis 0 or 1 or 2 (1 is by default)

Mode '0' disables this parameter

Mode '1' enables the parameter only on the final encode of the macroblock

Mode '2' enables the parameter for all mode decisions.

Zones- This parameter is very efficient and powerful for the videos sequences where the user needs specific performance and parameter changes in particular scenes or frames. With this parameter, the user can define most of the x264 options for any specific zone (also decided by user). The user can define the zones by mentioning '/' and putting the options in each zone as <startframe>, <endframe>, <options>.

Options for zones are as follows:

The user can only define one out of 'b' and 'q'. 'b' is a bit rate multiplier specific to every zone. It takes values in floats. User can define it as b=<float value>. 'q' provides the constant quantization value for a specific zone. It takes values as integers. User can define it as q=<integer value>. Other options of x264 which the user can use in zones are:

- ref=<integer>
- b-bias=<integer>
- scenecut=<integer>
- no-deblock
- deblock=<integer>:<integer>
- deadzone-intra=<integer>
- deadzone-inter=<integer>
- direct=<string>
- merange=<integer>
- nr=<integer>
- subme=<integer>
- trellis=<integer>
- (no-)chroma-me
- (no-)dct-decimate
- (no-)fast-pskip
- (no-)mixed-refs
- psy-rd=<float>:<float>

- me=<string>
- no-8x8dct
- b-pyramid=<string>
- crf=<float>

There are some limitations for applying above options on every zone. Zone options limitations are as follows:

1. Reference frames mentioned by parameter –ref cannot exceed in a zone.
2. Scenecut parameter cannot be turned ON or OFF but can be varied.
3. Motion estimation range cannot be re-specified what has been already defined initially.
4. Sub-pixel motion estimation complexity cannot be changed if it is defined as zero initially.

Partitions- Initially, frames get split into 16x16 blocks but this parameter in x264 allows the encoder as well as the user to define which partition is chosen for each frame/slice. It can vary from 16x16 to 4x4. The x264 available partitions are i8x8, i4x4, p8x8 (enables p16x8/p8x16), p4x4 (enables p8x4/p4x8), b8x8. The user can also select ‘all’ or ‘none’ as partitions.

Default: 'p8x8,b8x8,i8x8,i4x4'.

Weighted Prediction- x264 allows the encoder to use the weighted prediction for B and P frames by default. The user can disable the weighted prediction of P and B frames by following parameters:

--no-weightb 1 and --weightp 0 (by default, they are both enabled in x264).

Motion Estimation (ME) - As motion estimation uses multiple prediction modes and reference frames, it is the most time consuming and complex process in encoding. The different motion estimation methods provided by x264 are: diamond (DIA), hexagon (HEX), uneven multi-hexagon (UMH), successive elimination exhaustive search (ESA), and transform successive elimination exhaustive search (TESA).

Diamond motion estimation search: Diamond motion search or dia is the simplest motion estimation search available in x264. It either starts from the previous best known position available or starts from the best predicted value. It starts checking the motion vector in all four directions (up, down, left and right) for the best match available based on the MAD (mean of absolute differences) values. It repeats the process until it finds the best

match for the target motion vector. Figure 3.1 illustrates the diamond search pattern, where '0' is the best predicted value and '1' are the positions to try next.

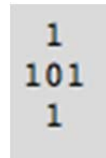


Figure 3.1. Diamond motion estimation search pattern.

User can define motion estimation search as diamond by:

--me dia.

Hexagon motion estimation search (Hex): Hexagon motion estimation search method or Hex works in a way similar to diamond search; however, it works until the range two instead of one for six surrounding points around the best predicted point from past. It provides better results and hence is not only more efficient than Dia but also fast. These benefits make it a better choice. Figure 3.2 illustrates the hexagon search pattern, where '0' is the best predicted point and '1's are the second in line for the iteration; if they are still not able to find the best value, it goes for '2's in range.

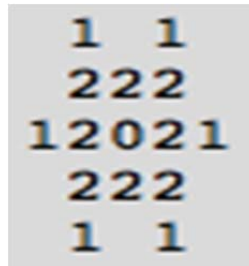


Figure 3.2. Illustrates the hexagon motion estimation search pattern.

User can define motion estimation search as hexagon by:

--me hex.

Uneven multilevel hexagon motion estimation search (UMH): Uneven multi-hex motion search method is considered slower than dia and hex. But the advantage of using this method is that it searches within very complex multi-hex motion estimation search patterns, which uncovers even the hidden matches and hence provides the best matches on the cost of speed as compared to previous methods. The user can control the search range in the UMH method, by controlling the search window by --merange parameter, which defines the search

window size. Figure 3.3 illustrates the search pattern. Uneven multi-hexagon has 3 different search patterns that are each run once. Then, it switches to the same pattern as hexagon.

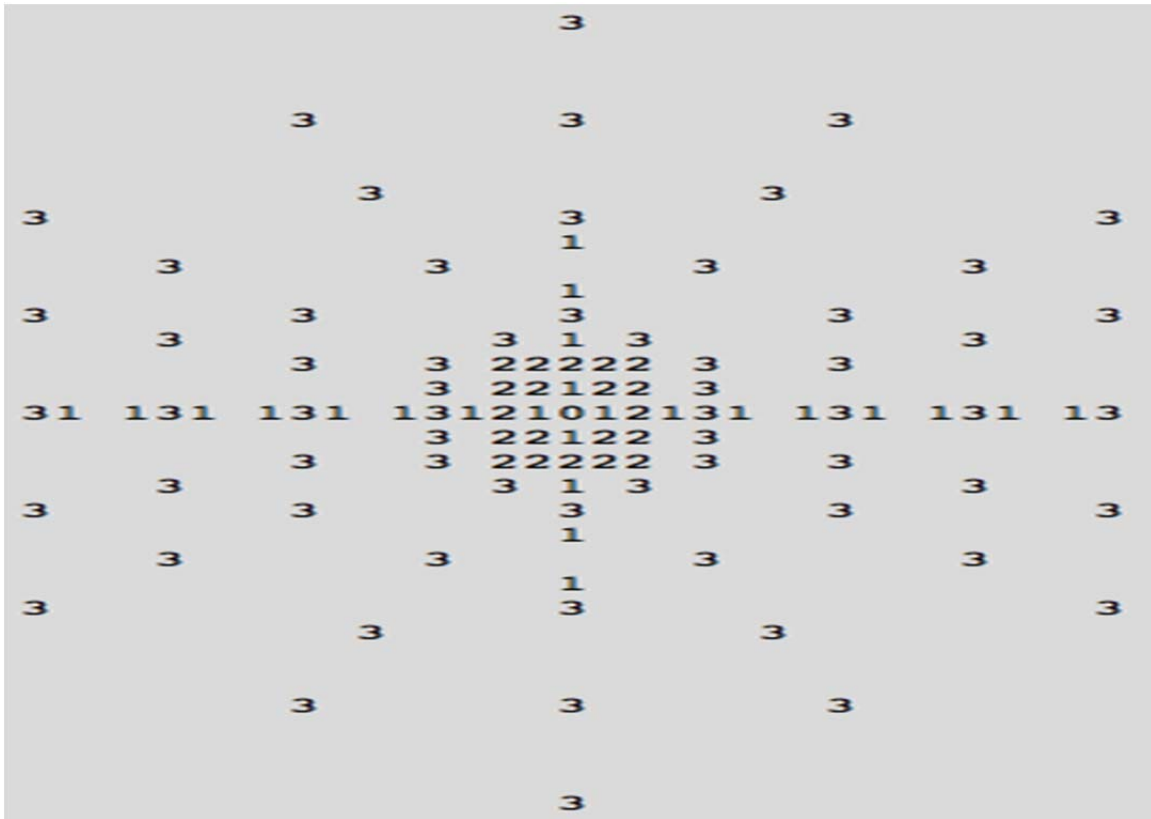


Figure 3.3. Replace exagon to uneven multi-hexagon.

User can define motion estimation search for umh by:

--me umh.

User can define motion estimation search window range for umh by:

--merange 16

Exhaustive search (ESA): Successive elimination exhaustive search method is the most optimized, intelligent and detailed search pattern used to find out the best match from the entire motion search range, which is defined by a user defined --merange parameter. It tries to search for every single motion vector in the range mentioned with the maximum speed possible. Because of its highly detailed and exhaustive search, it is a slower method than umh even though the difference in quality is not large. Figure 3.4 illustrates the exhaustive search pattern.

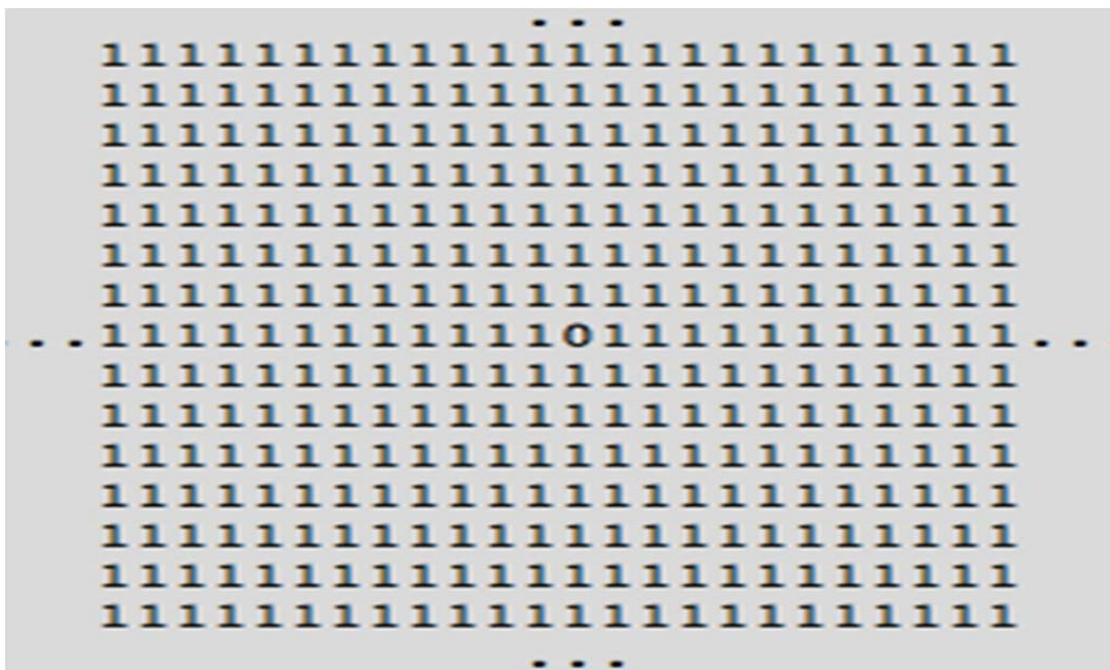


Figure 3.4. Illustrates the exhaustive search pattern.

User can define motion estimation search for esa by:

--me esa.

User can define motion estimation search window range for esa by:

--merange 32

Subpixel estimation- Subpixel estimation or '--subme' sets the subpixel estimation complexity for encoding. In x264, '--subme' is the parameter by which a user can enable rate distortion optimization (RDO) for mode decisions [15], motion vector and intra prediction modes. The user can define the level of rate distortion optimization needed for each different case of the experiment by defining the subme parameter's value. RDO becomes enabled after level 5. Different levels of subpixel motion estimation are as follows:

1. fullpel only
2. QPel SAD 1 iteration
3. QPel SATD 2 iterations
4. HPel on MB then QPel
5. Always QPel
6. Multi QPel + bi-directional motion estimation
7. RD on I/P frames
8. RD on all frames

9. RD refinement on I/P frames
10. RD refinement on all frames
11. QP-RD (requires `-trellis = 2`, `--aq-mode > 0`)

3.2 FFMPEG FEATURES AND PARAMETER SET

FFMPEG [16] is one of the fastest audio-video codecs with the capability to grab live streams as well. The general syntax to use FFMPEG for encoding or decoding is:

FFMPEG [global options] [[infile options][`-i infile`]]... {[outfile options] *outfile*}.

Input/Output- This option is to define the input or output video or audio file. It is defined in command line as: `-i [file name]` for input and `-y [file name]` for output file. The option `-y [file name]` overwrites the output file over previous one.

Format- This option is to define the format of video. It is defined in command line as: `-f [format]`. For example `-f rawvideo`.

Pixel Format- This option is to define pixel format for video. It is defined in command line as: `-pix_fmt [format]` for example `-pix_fmt yuv420p`.

Frame size- This option is to define the frame size for video. It is defined in command line as: `-s [frame size]`, for example `-s 352x288` for cif videos.

Frame rate- This option is to define frame rate for a given video. It is defined on command line as: `-r [frame rate]`, for example `-r 25`.

Pass- This option is to define the pass number for a given video. It is defined in command line as: `-pass [n]` for example `FFMPEG -i xxx.mov -pass 1 -f rawvideo -y /dev/null`.

RTP mode – This option is used to send the encoded stream using RTP mode to some other destination. In this mode, multiplexing is done after encoding and the de-multiplexing is done before decoding of the stream at the destination. Command line is used to send encoded stream via RTP mode as follows:

FFMPEG `-i [input.264] -vcodec [codec] -f rtp rtp://172.18.41.122:1000 [ip address]`.

As FFMPEG uses x264 libraries for H.264 encoding, mapping is needed for x264 commands for FFMPEG. The user can define x264 parameters from the FFMPEG command line but for that, 'x264' should be mentioned in the command line before putting x264 commands and ':' should be used before the next x264 command. For example:

`'FFMPEG -i [input.yuv] -pass 1 -x264opts slice-max-size=300:merange=5:keyint=20 -y out.264'`

Error concealment – The parameter that defines error concealment in FFMPEG command line is `'-ec bit_mask'`. The `bit_mask` is a bit mask of the following values:

`'1'` – `FF_EC_GUESS_MVS` (default=enable)

`'2'` – `FF_EC_DEBLOCK` (default=enable)

Error concealment schemes work in FFMPEG by checking and determining which parts of slices in a given frame are corrupted due to errors [17]. The code discards all data after error and also some data before error within a slice. After the discarding is done, the code guesses (based on the undamaged parts of slice and the past frame) whether concealment is better from the last frame or from the neighborhood (spatial). Based on this decision, it decides which macroblocks are unknown, lost or corrupted. After this, it estimates the motion vectors for all non-intra macroblocks, those have damaged motion vectors based on their neighboring blocks. After all these steps, all the damaged parts are passed through a deblocking filter to reduce the artifacts in the end.

x264 parameter mapping [18] for FFMPEG is as follows-

`--keyint <integer> (x264)`

`-g <integer> (FFMPEG)` – This option defines the GOP size in FFMPEG which is defined by the key frame interval value in x264. This is the distance between two key frames (i.e., IDR/I frames). A large GOP size gives more compression but finding scene change etc. becomes more complex. The default value for this option is 250.

`--min-keyint <integer> (x264)`

`-keyint_min <integer> (FFMPEG)` – This parameter defines the minimum GOP length, the minimum distance between the key frames. The default value for this parameter is 25.

`--scenecut <integer> (x264)`

`-sc_threshold <integer> (FFMPEG)` – As already explained above, this parameter adjusts the sensitivity of scenecut detection in x264. The default value is 40.

`--bframes <integer> (x264)`

`-bf <integer> (FFMPEG)` – This parameter controls the number of consecutive B frames. The maximum number allowed is 16 and the default value for this parameter is also 16.

`--b-adapt <integer> (x264)`

`-b_strategy <integer> (FFMPEG)` – This parameter adaptively decides the best number of B frames to be used in encoding for a given video sequence.

`--b-pyramid (x264)`

`-flags2 +bpyramid (FFMPEG)` – This parameter starts keeping and using B frames for reference in a decoded picture buffer.

`--ref <integer> (x264)`

`-ref <integer> (FFMPEG)` – This parameter maps the number of reference frames in FFMPEG for a given video sequence. The default value is 6.

`--no-deblock (x264)`

`-flags -loop (FFMPEG)` – This parameter disables the loop filter for encoding. The default value is `-flags +loop` (enable loop filter).

`--deblock <alpha:beta> (x264)`

`-deblockalpha <integer> (FFMPEG)`

`-deblockbeta <integer> (FFMPEG)` – Deblocking filter is one of the major feature in h.264 and this parameter enables it, which helps in removing blocking artifacts. To enable this parameter, the configuration must have `-flags +loop` (enable loop filter).

Rate control:

`--qp <integer> (x264)`

`-cqp <integer> (FFMPEG)` – This parameter gives constant quantizer mode. In reality, the quantizer value for B and I frames comes as different from P frames in this mode as well.

`--bitrate <integer> (x264)`

`-b <integer> (FFMPEG)` – This parameter defines the bitrate mode and gives the target bitrate to the encoder to achieve. To achieve a target bitrate mentioned by this parameter, the user should use 2-pass mode.

`--ratetol <float> (x264)`

`-bt <float> (FFMPEG)` – This parameter allows the variance of average bitrate.

--pass <1,2,3> (x264)

-pass <1,2,3> (FFMPEG) – This parameter is to define the pass in FFMPEG and should always be used with any target bitrate mentioned for given video sequence as *-b <integer>*.

--partitions <string> (x264)

p8x8 , p4x4, p8x8, i8x8, i4x4

-partitions <string> (FFMPEG)

+partp8x8, +partp4x4, +partb8x8, +parti8x8, +parti4x4

This parameter helps in choosing the macroblock partition in h.264 for I, P and B macroblocks.

--weightb (x264)

-flags2 +wpred (FFMPEG) – This parameter allows B frames to use weighted prediction for a given video sequence.

--me <dia,hex,umh,esa> (x264)

-me_method <epzs, hex, umh, full> (FFMPEG) – These search methods are already explained in Section 2.1.1. This parameter enables different motion estimation methods in FFMPEG for a given video sequence.

--merange <integer> (x264)

-me_range <integer> (FFMPEG) - This parameter controls the range for motion search.

--subme <integer> (x264)

-subq <integer> (FFMPEG) – The function of this parameter is also described above.

This parameter is to define the subpixel motion estimation mode [19]. This is the mode that defines where the user can use RDO for a given video sequence.

--mixed-refs (x264)

-flags2 +mixed_refs (FFMPEG) – This parameter allows the p8x8 block to select different references for each p8x8 block.

CHAPTER 4

DESCRIPTION OF SYSTEM SET UP

4.1 INTRODUCTION

In this chapter, the simulation setup for H.264 encoding is described. Here, the whole set up for two different codecs is described to get a clear understanding. In this section, we describe the various parameters and their values used for simulations, software versions of both codecs and other specifications/configurations used in the experiments.

4.2 SIMULATION ENVIRONMENT FOR JM

The simulation environment in both codecs JM [10] and FFMPEG is kept at such a level where we can compare the features and results in terms of various parameters [20]. The summarization of simulation environment in JM is as follows:

- Software version: JM 14.2 with encoding and the modified changes for decoding as it has better rate control.
- Rate control was used with four different bitrates (256kbps, 512kbps, 768kbps and 1024kbps) with R-D optimization enabled. However, in the case of the BUS sequence, which has high motion, the results with rate control are to be taken with caution since it was difficult to fit the sequence and the packetization into the bit rate budget.
- For all simulations, concealment was done using motion copy due to its superior performance.
- FMO and data partitioning features are disabled in JM.
- For more details please see Appendix A.

4.2.1 Encoder Parameter

For most of the cases, the following parameters were used.

- | | |
|--------------------------------|--|
| • Bit stream mode | : Annex B |
| • Motion vector resolution | : 1/4 pel |
| • Number of Reference Pictures | : 6 |
| • B-frames | : Used |
| • I frame rate | : 10 (because B frames are enabled, the frequency of I frame is twice of this) |
| • IDR rate | : 0 (First frame is still an IDR frame) |

- Profile : High
- GOP structure : IBPBPBP in Frame mode
- RD optimization : Used
- Constrained intra prediction: enabled
- Slice mode : Fixed bytes in slice
- Slice size : 150 bytes
- Rate control : Enabled
- Motion estimation search mode : Full search

4.2.2 Decoder Parameter

For fair comparison with FFMPEG, the error concealment algorithm [21] in JM14.2 was not used; instead, the macroblock level copy paste concealment was used. [22]

- POC Scale : 2 (B frames POC increase by value of 2 for every frame)
- Error concealment : 2 (Motion copy is used in all our simulations because it performs better than frame copy)
- Reference POC : 4 (I, P frame POC increases by 4 for every frame)
- POC Gap : 2 (Because of the coding pattern in IBPBPBP, and as B frames have POC scale of 2 and P frames have POC scale of 4, one can observe that the order of POC will be 0,2,4,6,8 etc. and resets whenever an IDR frame is received)

4.3 SIMULATION ENVIRONMENT FOR FFMPEG

The simulation environment is at the same level as JM in terms of features used to generate results. The summarization of the simulation environment in FFMPEG is as follows:

- Software version: the x264 encoder version used for all encoding in FFMPEG is “r2008m 4c552d8” and the FFMPEG decoder version for all decoding is “master-git-N-30860-g83f9bc8.” [23]
- Rate control was used with four different bitrates (256kbps, 512kbps, 768kbps and 1024kbps) with R-D optimization enabled. To achieve the target bitrates, the multi-pass encoding mode was selected for final comparison results.
- For all simulation purposes in FFMPEG concealment was done mostly by motion copy. However, frame copy was not disabled either.
- FMO and Data partitioning is not present to support H.264 coding standard.
- For more details please see Appendix B.

4.3.1 Encoder Parameters

X264 encoder is used by FFMPEG for all encoding. For most of the cases, the following parameters were used.

- Bit stream mode (by default) : Annex B
- Motion vector resolution : 1/4 pel
- No. of Reference Pictures (-refs) : 6
- B-frames (-bframes) : Used
- I frame rate (-keyint) : 20
- IDR rate : 0 (First frame is still an IDR frame)
- Profile (-profile) : High
- GOP structure : IBPBPBP in Frame mode
- RD optimization (-subme) : Used
- Constrained intra prediction : enabled
- Slice mode : Fixed bytes in slice
- Slice size (-slice-max-size) : 150 bytes
- Rate control (multipass(n), trellis) : Enabled
- Motion estimation search mode (-me_method) : Exhaustive full search

4.3.2 Decoder Parameter

FFMPEG decoder is used for all the decoding in these experiments [11]. For error concealment, the motion copy was used; however, frame copy also remained enabled in the code.

CHAPTER 5

EXPERIMENTAL EVALUATION OF FFMPEG AND JM CODEC

5.1 INTRODUCTION

In this chapter, we discuss simulation results to evaluate the performance of the FFMPEG codec and compare it with that of JM. We did many experiments on FFMPEG codec with and without errors, and calculated parameters like PSNR, VQM, encoding time, and decoding time to analyze its performance for different bitrates (i.e., 256kbps, 512kbps, 768kbps and 1024 kbps) and video streams. We also studied the impact of various features of FFMPEG on its performance.

5.2 EXPERIMENT AND COMPARISON WITHOUT LOSSES

Tables 5.1-5.6 show the performance of the single and two-pass FFMPEG and JM codecs in terms of their bit rate, PSNR, VQM, encoding and decoding time [24]. We have used the CIF (352x288 pixels, 24 bits/pixel, RGB) Bus and News test video sequences to conduct the comparison at different bit rates and for different NAL sizes.

Tables 5.1, 5.2 and 5.3 show results for the BUS video sequence for single pass FFMPEG, two pass FFMPEG, and JM codec, respectively. Similarly, Tables 5.4, 5.5 and 5.6 show results for News video sequence for single pass FFMPEG, two pass FFMPEG, and JM codec, respectively. We observe that the two pass FFMPEG and JM achieve tighter rate control of the encoding bit rate as compared to the single pass FFMPEG codec. Single pass FFMPEG is overshooting about 4 – 46 kbps from target bitrate while two pass FFMPEG is always under the target within a range of 2 – 28 kbps. JM varies from 0 – 24 kbps from target bitrate.

The encoding and decoding time is better for the FFMPEG single pass and the encoding time in two-pass FFMPEG is twice that of the single pass scheme. JM codec has a very large encoding time (~1700 sec) because its code is not optimized for fast execution. In

Table 5.1. Results of BUS Sequence with Single Pass and Without Errors (FFMPEG)

Bus, GOP=20; FFMPEG- n=1 (single pass encoding)					
Target - 128Kbps					
Slice-size	Achieved bitrate	PSNR	VQM	Encoding time(sec)	Decoding time (sec)
150	131.7	24.14	5.10	10	0.5
300	132.2	24.49	4.91		
500	132.1	24.61	4.87		
Target bitrate- 256 Kbps					
Slice-size	Achieved bitrate	PSNR	VQM		
150	265.3	27.17	3.69	10	0.5
300	265.2	27.55	3.54		
500	265.1	27.71	3.49		
Target bitrate- 512 Kbps					
Slice-size	Achieved bitrate	PSNR	VQM		
150	534.3	30.52	2.54	10	0.5
300	533.8	30.96	2.43		
500	534.4	31.15	2.38		
Target bitrate- 768 Kbps					
Slice-size	Achieved bitrate	PSNR	VQM		
150	803.3	32.61	2.02	10	0.5
300	802.5	33.06	1.92		
500	802.1	33.26	1.88		
Target bitrate- 1024 Kbps					
Slice-size	Achieved bitrate	PSNR	VQM		
150	1069.8	34.12	1.71	10	0.5
300	1068.9	34.61	1.62		
500	1068.6	34.79	1.59		

terms of PSNR and VQM performance, we observe that both versions of FFMPEG codec achieve slightly better video quality than JM.

Test scenario: Bus video sequence, FFMPEG codec; 30fps; GOP size: 20 frames; Bit Rates: 128, 256, 512, 768, and 1024 kbps; NAL size: 150, 300 and 500 bytes.

Test scenario 2: News video sequence encoded using the single pass FFMPEG codec; the parameters are same as in Scenario 1. Table 5.4 illustrates the results of this experiment.

Table 5.2. Results of BUS Sequence with Two Pass and Without Errors (FFMPEG)

Bus, GOP=20; FFMPEG- n=2 (Two pass encoding)					
Target - 128Kbps					
Slice-size	Achieved bitrate	PSNR	VQM	Encoding time(sec)	Decoding time (sec)
150	123.3	23.93	5.24	20	0.5
300	123.4	24.31	5.05		
500	122.9	24.42	4.99		
Target bitrate- 256 Kbps				20	0.5
Slice-size	Achieved bitrate	PSNR	VQM		
150	247.5	26.91	3.81		
300	247.3	27.3	3.64	20	0.5
500	247.9	27.45	3.57		
Target bitrate- 512 Kbps				20	0.5
Slice-size	Achieved bitrate	PSNR	VQM		
150	498.2	30.19	2.63		
300	500.1	30.65	2.51	20	0.5
500	499.4	30.83	2.45		
Target bitrate- 768 Kbps				20	0.5
Slice-size	Achieved bitrate	PSNR	VQM		
150	748.1	32.27	2.09		
300	752.6	32.75	1.98	20	0.5
500	756.3	32.96	1.94		
Target bitrate- 1024 Kbps				20	0.5
Slice-size	Achieved bitrate	PSNR	VQM		
150	998	33.76	1.77		
300	1008.4	34.29	1.67	20	0.5
500	1006	34.47	1.64		

5.3 EXPERIMENTS AND COMPARISON WITH LOSSES

In these experiments, we introduced a different percentage of errors in encoded video streams to study the performance of FFMPEG and JM, including their error concealment schemes.

Table 5.3. Results of BUS Sequence Without Errors (JM)

Bus, GOP=20 JM					
Target - 128Kbps					
Slice-size	Achieved bitrate	PSNR	VQM	Encoding time(sec)	Decoding time (sec)
150	130	23.22	4.036	1700	1.4
300	132	23.4	4.02	2005	1.8
500	132	23.91	4.03	1699	1.8
Target bitrate- 256 Kbps					
Slice-size	Achieved bitrate	PSNR	VQM		
150	257	26.83	3.969	2101	1.21
300	256	27.06	3.874	1712	1.24
500	256	27.16	3.822	1717	1.19
Target bitrate- 512 Kbps					
Slice-size	Achieved bitrate	PSNR	VQM		
150	512	29.84	2.89	1789	3.5
300	514	30.17	2.785	1938	1.5
500	508	30.28	2.754	2498	1.4
Target bitrate- 768 Kbps					
Slice-size	Achieved bitrate	PSNR	VQM		
150	765	31.47	2.343	1812	4.3
300	777	31.85	2.26	2214	1.8
500	782	32.11	2.203	2639	1.6
Target bitrate- 1024 Kbps					
Slice-size	Achieved bitrate	PSNR	VQM		
150	1036	33.27	1.933	2711	2.6
300	1047	33.79	1.837	2604	2.2
500	1038	33.86	1.833	2320	1.9

5.3.1 Enhancements to FFMPEG

The default FFMPEG setup only accepts % bit losses and did not allow the slice or NALU losses [25]. We developed a separate module to introduce the slice or NALU losses in FFMPEG. In this module, we read the H.264 bitstream and introduce slice/NALU losses based on user inputs. For this, we first search for the 24-bit start code (0x000001) in the bitstream, which is followed by the NAL unit type (i.e. sequence parameter, picture

Table 5.4. Results of News Sequence with Single Pass and Without Errors (FFMPEG)

News, GOP=20 FFMPEG- n=1 (single pass encoding)					
Target - 128Kbps					
Slice-size	Achieved bitrate	PSNR	VQM	Encoding time(sec)	Decoding time (sec)
150	138.6	34.44	1.68	10	0.5
300	138	33.25	1.61		
500	137.6	33.44	1.57		
Target bitrate- 256 Kbps				10	0.5
Slice-size	Achieved bitrate	PSNR	VQM		
150	276	37.13	1.1		
300	275.5	37.59	1.04		
500	275.3	37.79	1.02		
Target bitrate- 512 Kbps				10	0.5
Slice-size	Achieved bitrate	PSNR	VQM		
150	549.5	41.06	0.74		
300	547.8	41.44	0.71		
500	547.5	41.62	0.7		
Target bitrate- 768 Kbps				10	0.5
Slice-size	Achieved bitrate	PSNR	VQM		
150	820.9	43.04	0.6		
300	818.3	43.41	0.58		
500	817.6	43.55	0.57		
Target bitrate- 1024 Kbps				10	0.5
Slice-size	Achieved bitrate	PSNR	VQM		
150	1090.3	44.34	0.52		
300	1086.9	44.74	0.5		
500	1086	44.89	0.49		

parameter) and NALU payload data. Based on the above information and the user inputs, our code can introduce any loss pattern as explained in Annexure H.264- FFMPEG and JM.

Another problem was that FFMPEG code was not able to handle the loss of the first slice or its header from a frame. This was because the parser (i.e. libavcodec/h264_parser.c) detects access unit boundaries based on the first slice and the loss of that slice was responsible for failing the code. We resolved this issue by providing access unit boundaries in the picture header instead of the slice header. The code was updated in the next FFMPEG patch mentioned in Appendix B.

Table 5.5. Results of News Sequence with Two Pass and Without Errors (FFMPEG)

News, GOP=20							
FFMPEG- n=2 (Two pass encoding)							
Target - 128Kbps							
Slice-size	Achieved bitrate	PSNR	VQM	Encoding time(sec)	Decoding time (sec)		
150	126.6	32.82	1.71	20	0.5		
300	126.7	33.33	1.62				
500	127.2	33.38	1.61				
Target bitrate- 256 Kbps							
Slice-size	Achieved bitrate	PSNR	VQM	20	0.5		
150	253.2	37.13	1.11				
300	254	37.67	1.04				
500	255.3	37.88	1.03	20	0.5		
Target bitrate- 512 Kbps							
Slice-size	Achieved bitrate	PSNR	VQM				
150	507.4	41.02	0.75	20	0.5		
300	511	41.5	0.72				
500	510	41.66	0.7				
Target bitrate- 768 Kbps							
Slice-size	Achieved bitrate	PSNR	VQM	20	0.5		
150	765	42.98	0.61				
300	766.8	43.39	0.58				
500	768.7	43.56	0.57	20	0.5		
Target bitrate- 1024 Kbps							
Slice-size	Achieved bitrate	PSNR	VQM				
150	1018	44.22	0.53	20	0.5		
300	1023.5	44.66	0.51				
500	1023.9	44.81	0.5				

5.3.2 Results for Weighted Prediction and Trellis Parameter in FFMPEG

In this test case, the weighted prediction for P and B frames was enabled while trellis was disabled in order to study their effect on PSNR. Trellis was disabled for a fair comparison with JM as it is not present in JM. We randomly dropped 5% slices in FFMPEG and JM encoded video streams. Figure 5.1 illustrates how to introduce errors in FFMPEG. To

Table 5.6. Results of News Sequence Without Errors (JM)

News, GOP=20 JM					
Target - 128Kbps					
Slice-size	Achieved bitrate	PSNR	VQM	Encoding time(sec)	Decoding time (sec)
150	129	33.05	1.786	3266	1.9
300	128.5	33.3	1.733	4230	1.8
500	128	33.54	1.694	2232	1.8
Target bitrate- 256 Kbps					
Slice-size	Achieved bitrate	PSNR	VQM		
150	257	36.91	1.192	4232	2.3
300	257	37.27	1.15	3982	2.1
500	257	37.42	1.133	3864	2
Target bitrate- 512 Kbps					
Slice-size	Achieved bitrate	PSNR	VQM		
150	517	40.47	0.821	5240	3.4
300	517	40.75	0.794	4238	2.8
500	518	40.88	0.785	5088	2.6
Target bitrate- 768 Kbps					
Slice-size	Achieved bitrate	PSNR	VQM		
150	777	43.32	0.624	3544	3.8
300	775	43.64	0.603	5065	3.3
500	778	43.77	0.597	4029	3
Target bitrate- 1024 Kbps					
Slice-size	Achieved bitrate	PSNR	VQM		
150	1023	44.52	0.549	3581	4.9
300	1030	44.9	0.53	4886	3.6
500	1034	45.04	0.519	3477	3.4

introduce errors in FFMPEG, the user must have access to the encoded video file and then create text files including corrupted/lost slice numbers.

For our study, we used CIF Foreman video sequence encoded by JM and FFMPEG codecs at 30fps, 20 frame GOP size, 512 kbps bit rate and 150 bytes slice size. Figure 5.2 illustrates the average PSNR for 5% random slice loss for 10 cases, each with a different error seed for both codecs. We observe the difference in PSNR values in both codecs. It

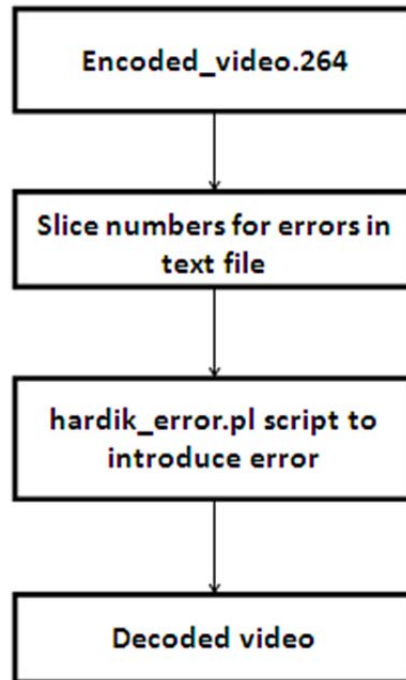


Figure 5.1. Illustration of how to introduce slice errors in FFMPEG coder.

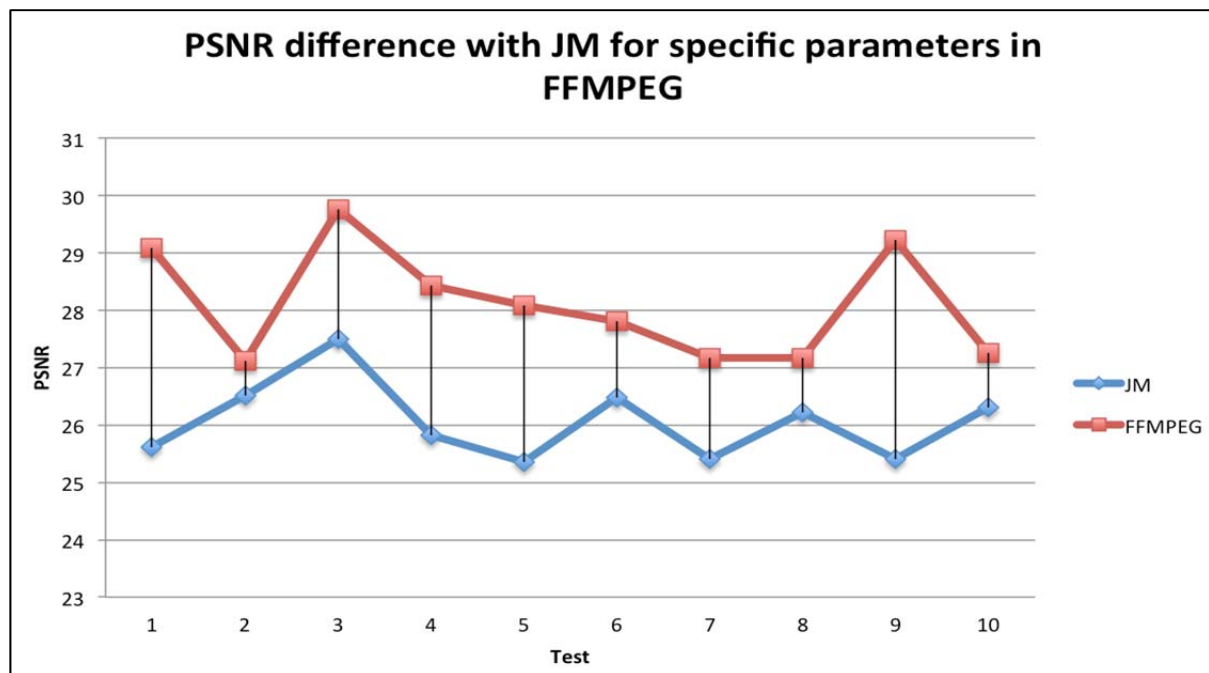


Figure 5.2. Average PSNR comparison of JM and FFMPEG by enabling weighted prediction and disabling trellis parameter in FFMPEG.

shows the importance of the weighted prediction mode. The FFMPEG codec achieves better PSNR values than JM. The PSNR values without any losses in FFMPEG and JM are 36.28 dB and 36.44 dB, respectively.

5.3.3 Results for Different Loss Patterns in FFMPEG vs. JM

In this experiment, we studied the PSNR performance of both codecs by introducing 2% and 5% random slice losses in the encoded streams of CIF size Bus and Foreman test video sequences [26]. For more details of the process, see Appendix C. We used the encoder parameters described in previous sections at 150 byte slice size and three bitrates - 256 kbps, 512 kbps and 1024 kbps for Bus and 512 kbps and 1024 kbps for Foreman. Tables 5.7 and 5.8 show the average PSNR achieved by both codecs for 10 cases, each with a different error seed for both codecs. We observe that the FFMPEG has better PSNR performance than JM for all the cases. With 2% random loss, the mean PSNR in FFMPEG is better by 0.04 – 2.17 dB for both video sequences. With 5% random loss, the mean PSNR is better in FFMPEG by 0.14 – 2.09 db. Figure 5.3- 5.12 illustrate the per-frame and per-GOP PSNR performance of both codecs for test conditions for both video sequences. We observe that FFMPEG generally achieves higher PSNR than JM does for most of the frames.

Table 5.7. Results of Bus_CIF Sequence with 2% and 5% Errors in FFMPEG and JM

BUS							
Bitrate (kbps)	Error free PSNR (dB)		% Random Error	PSNR (dB)			
				JM		FFMPEG	
	JM	FFMPEG		Mean	Std. Dev.	Mean	Std. Dev.
256	28.03	26.92	2	24.78	0.432	24.82	0.495
512	30.94	30.19		25.92	1.032	27.08	0.589
1024	34.66	33.84		28.48	0.579	28.96	0.697
256	28.03	26.92	5	22.01	0.455	23	0.621
512	30.94	30.19		23.5	0.478	24.33	0.362
1024	34.66	33.84		25.3	0.372	25.65	0.366

Table 5.8. Results of Foreman_CIF Sequence with 2% and 5% Errors in FFMPEG and JM

FOREMAN							
Bitrate (kbps)	Error free PSNR (dB)		% Random Error	PSNR (dB)			
				JM		FFMPEG	
	JM	FFMP EG		Mean	Std. Dev.	Mean	Std. Dev.
512	36.44	36.45	2	28.69	1.02	30.86	1.23
1024	39.4	39.45		32.12	0.42	32.76	0.82
512	36.44	36.45	5	26.06	0.68	28.15	0.62
1024	39.4	39.45		28.94	0.28	29.08	0.67

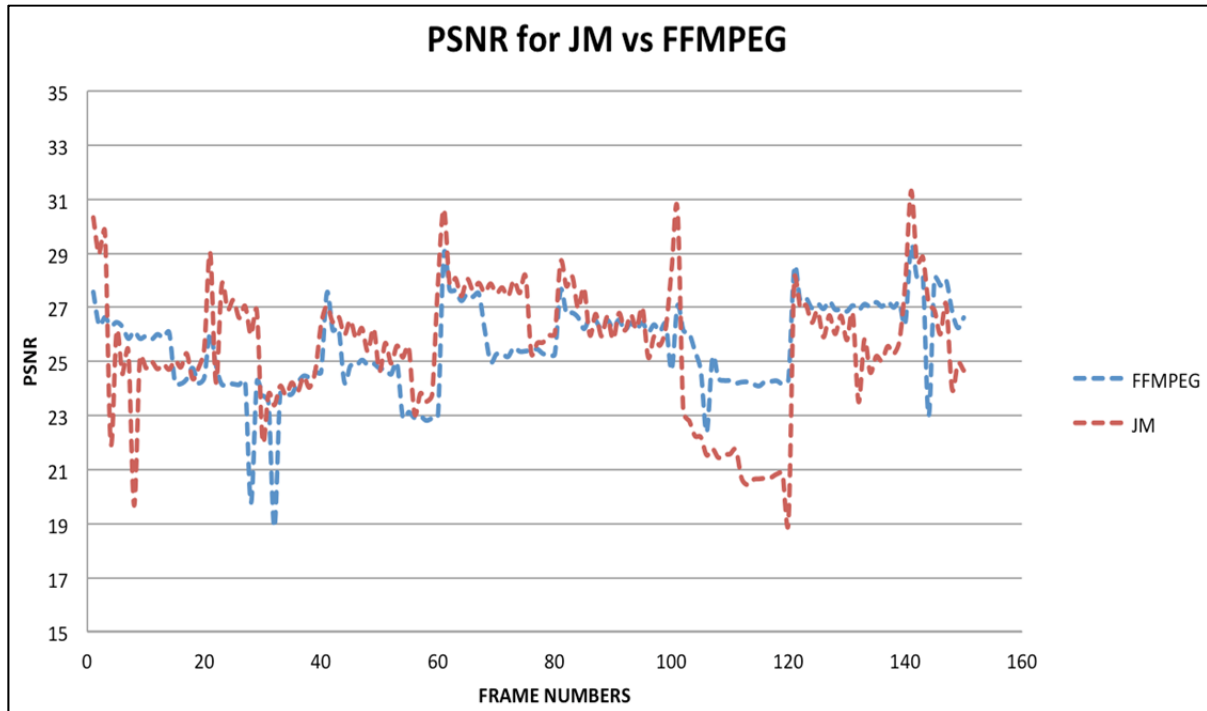


Figure 5.3. PSNR values of frames for JM vs FFMPEG with 2% slice loss for 256 kbps Bus sequence.

5.4 PRIORITY SCHEME IMPLEMENTED ON JM AND FFMPEG SLICES

This section covers the details of the priority scheme implemented on JM and FFMPEG slices. These slices priorities can be used for their more robust transmission against slice losses or packet drops. [27, 28]

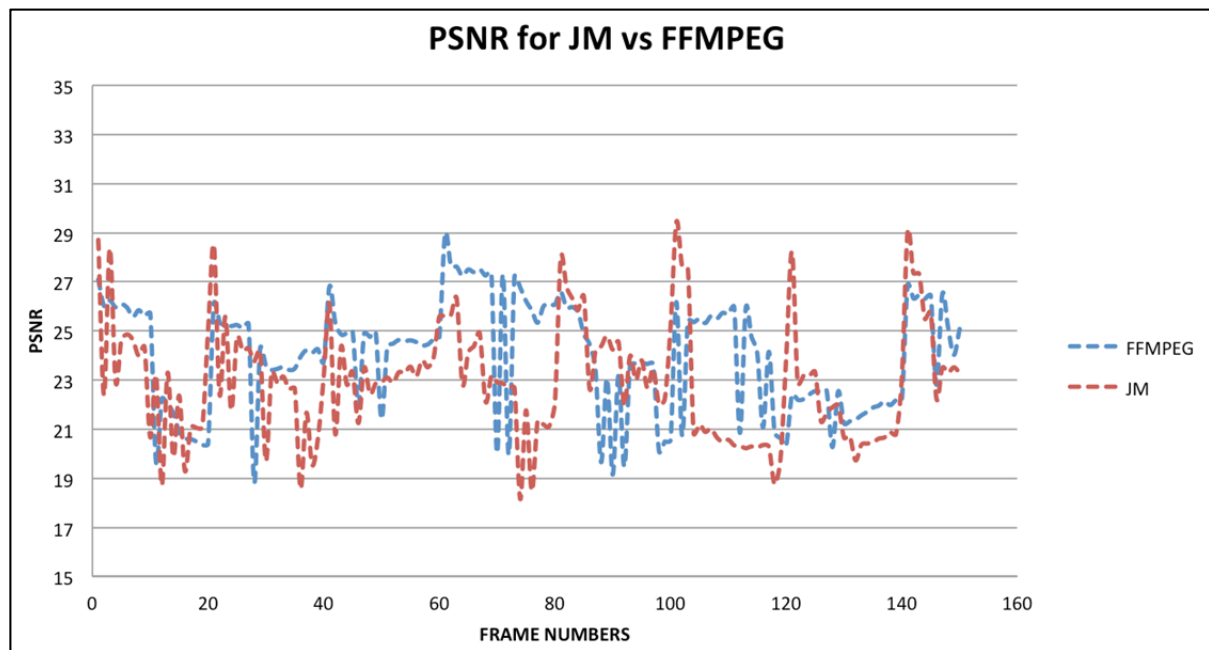


Figure 5.4. PSNR values of frames for JM vs FFMPEG with 5% slice loss for 256 kbps Bus sequence.

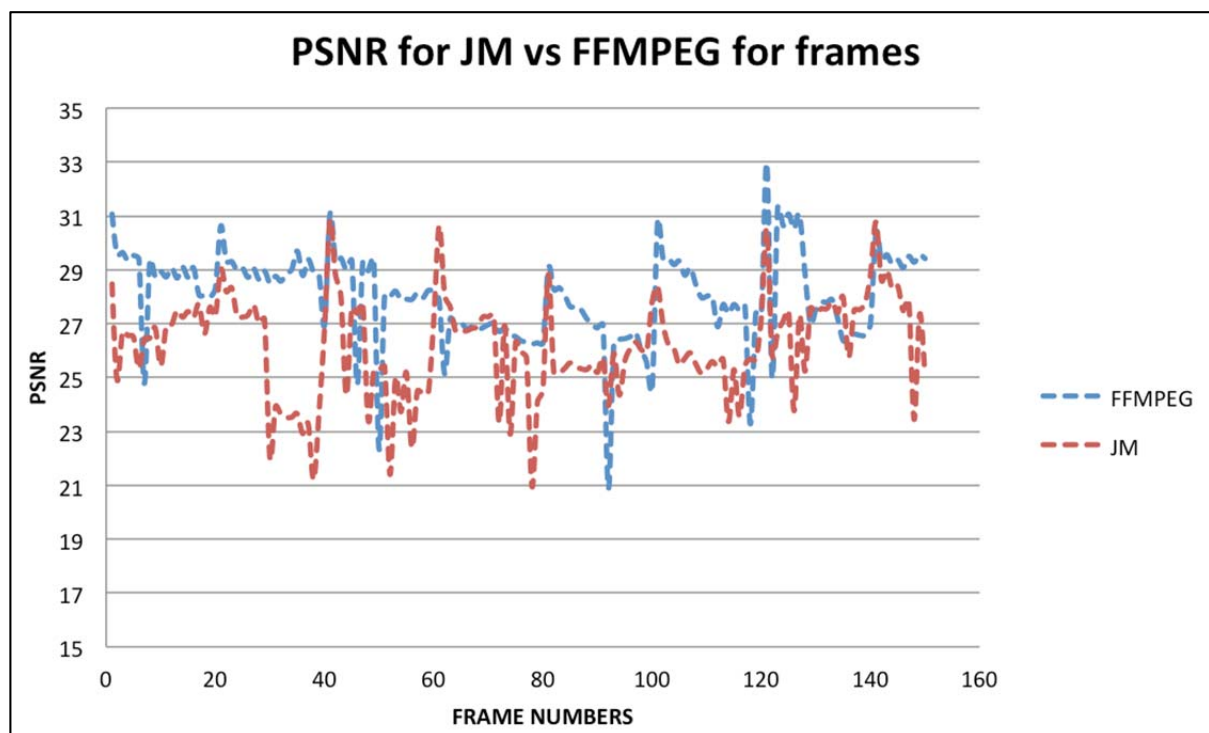


Figure 5.5. PSNR values of frames for JM vs FFMPEG with 2% slice loss for 512 kbps Bus sequence.

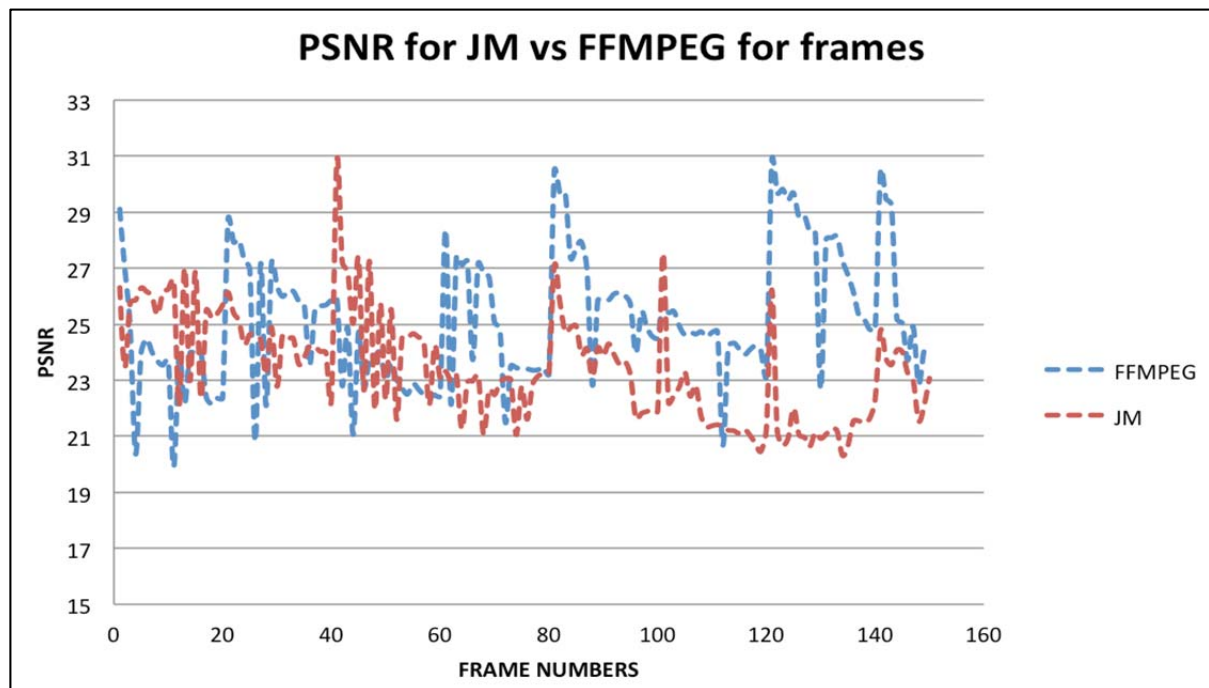


Figure 5.6. PSNR values of frames for JM vs FFMPEG with 5% slice loss for 512 kbps Bus sequence.

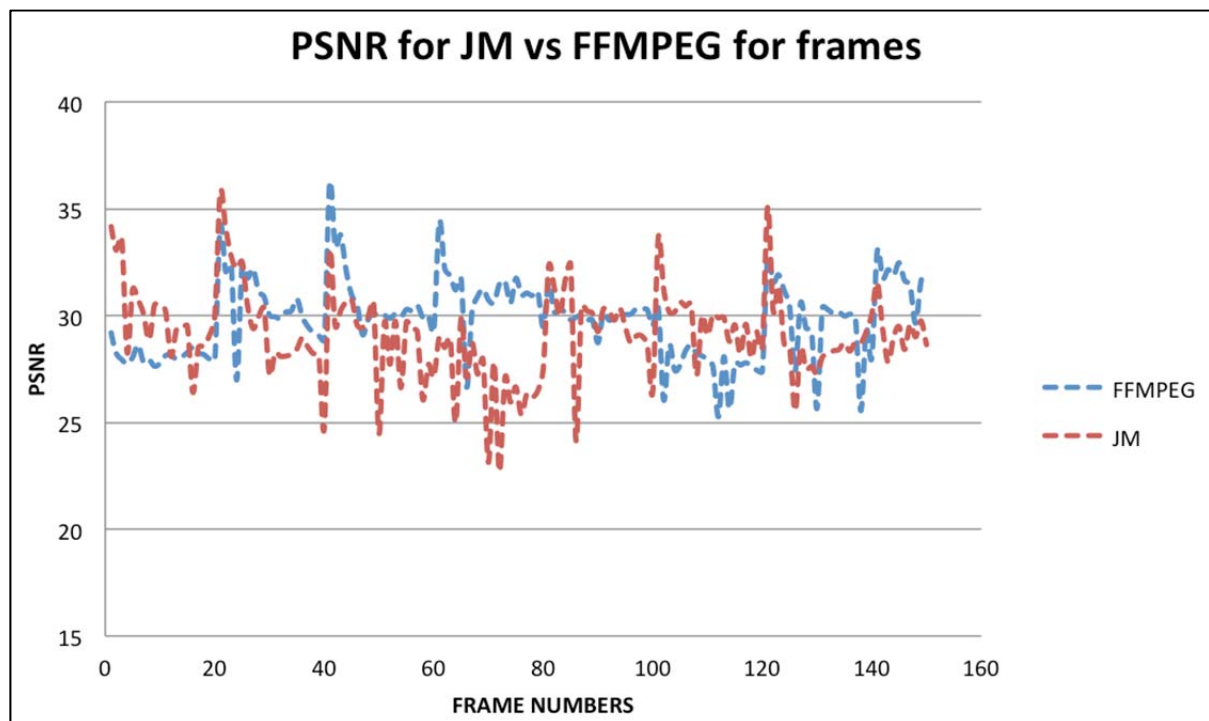


Figure 5.7. PSNR values of frames for JM vs FFMPEG with 2% slice loss for 1024 kbps Bus sequence.

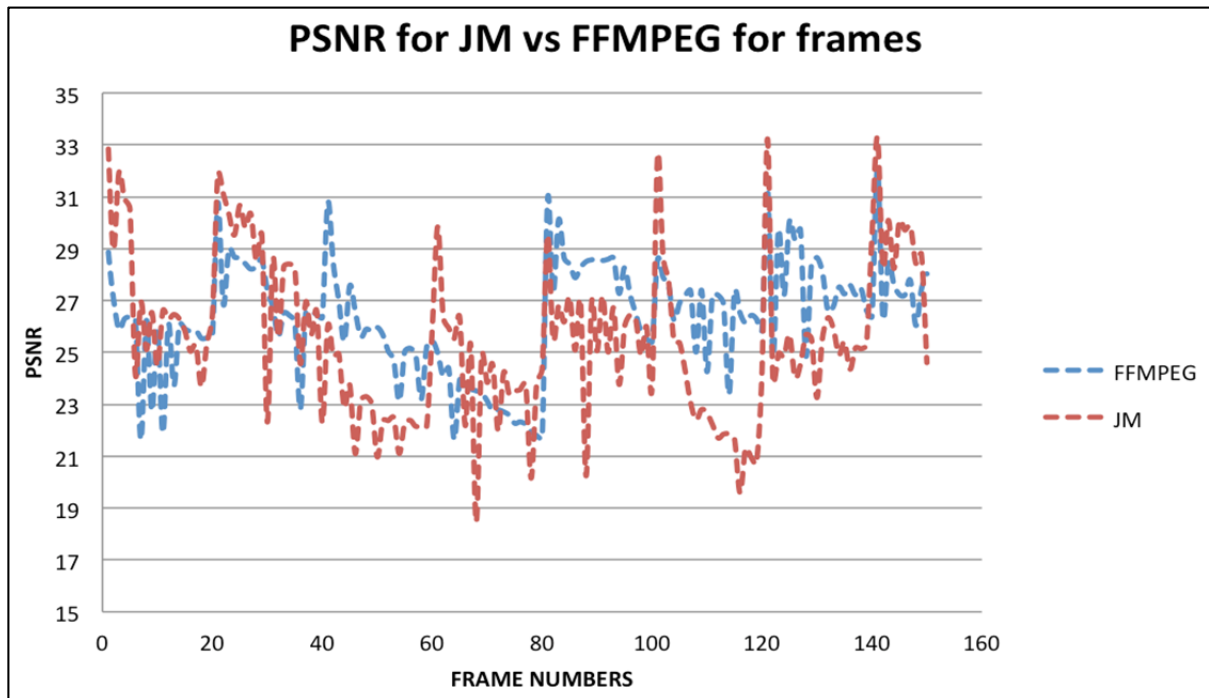


Figure 5.8. PSNR values of frames for JM vs FFMPEG with 5% slice loss for 1024 kbps Bus sequence.

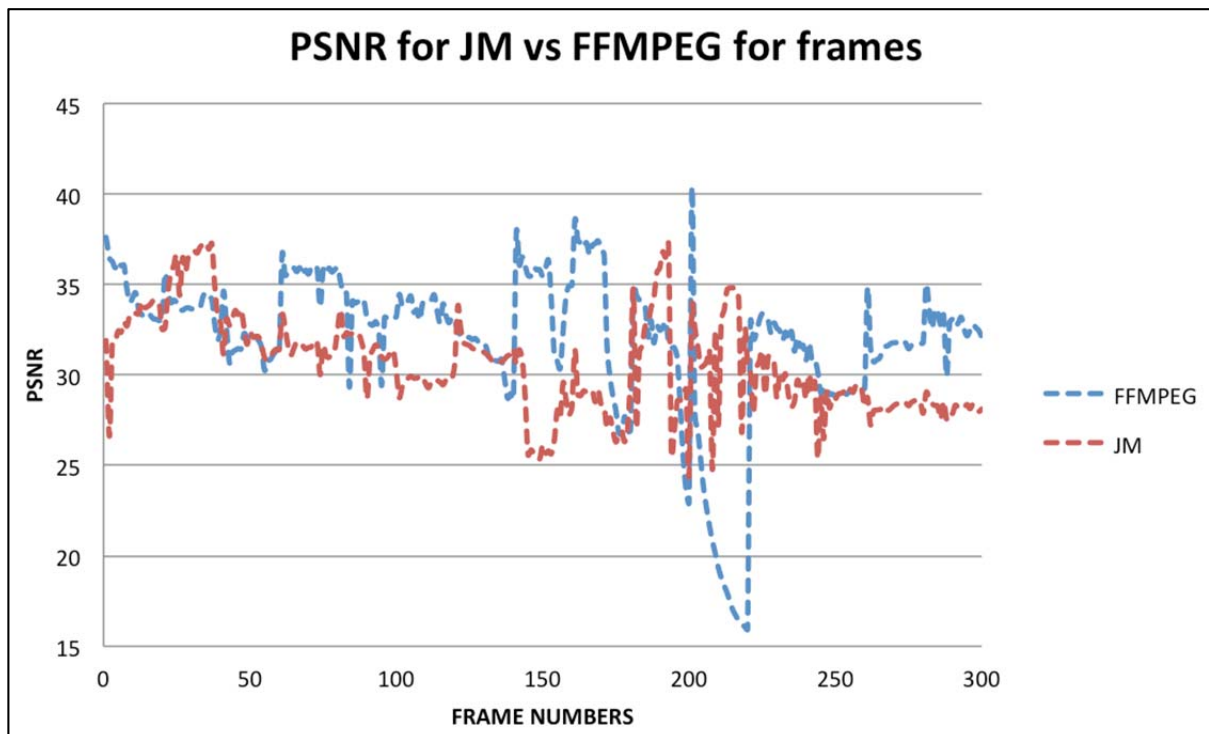


Figure 5.9. PSNR values of frames for JM vs FFMPEG with 2% slice loss for 512 kbps Foreman sequence.

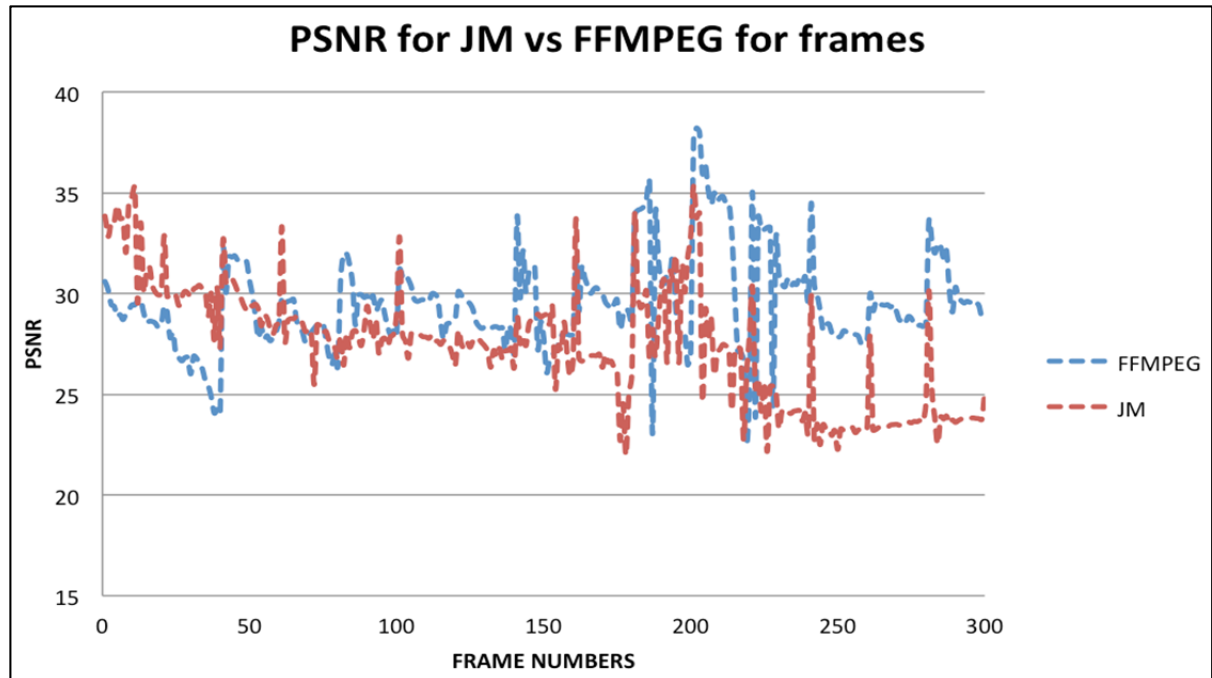


Figure 5.10. PSNR values of frames for JM vs FFMPEG with 5% slice loss for 512 kbps Foreman sequence.

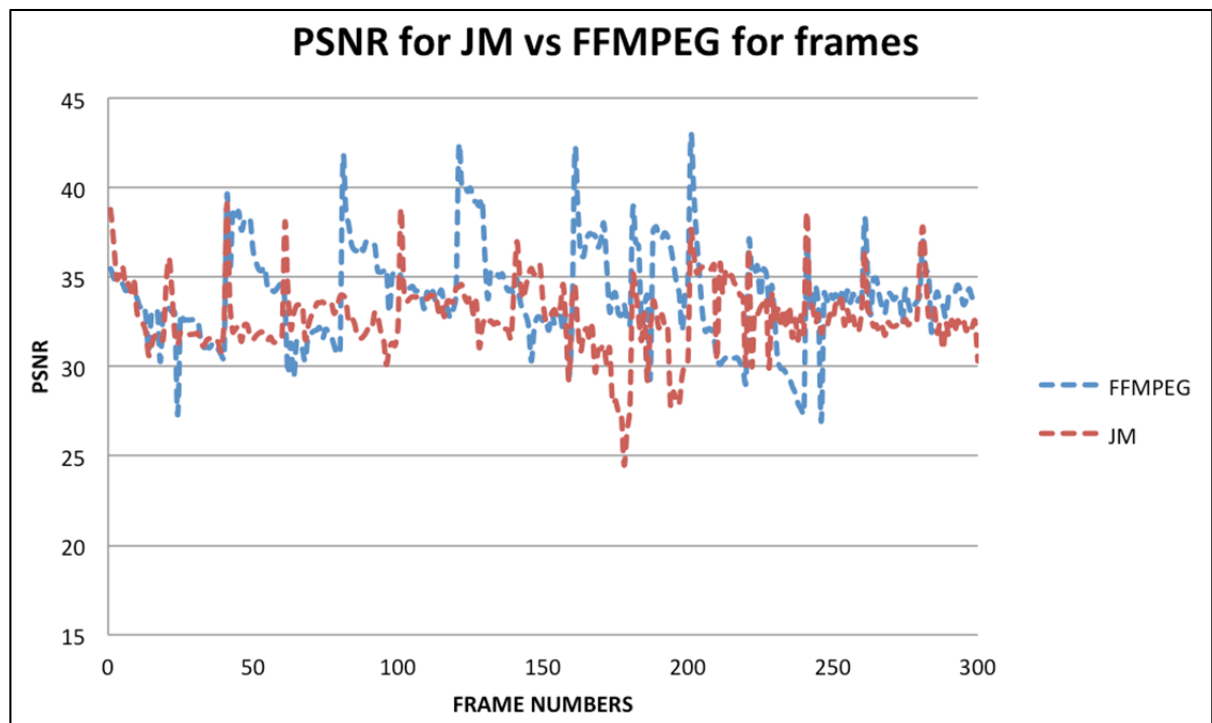


Figure 5.11. PSNR values of frames for JM vs FFMPEG with 2% slice loss for 1024 kbps Foreman sequence.

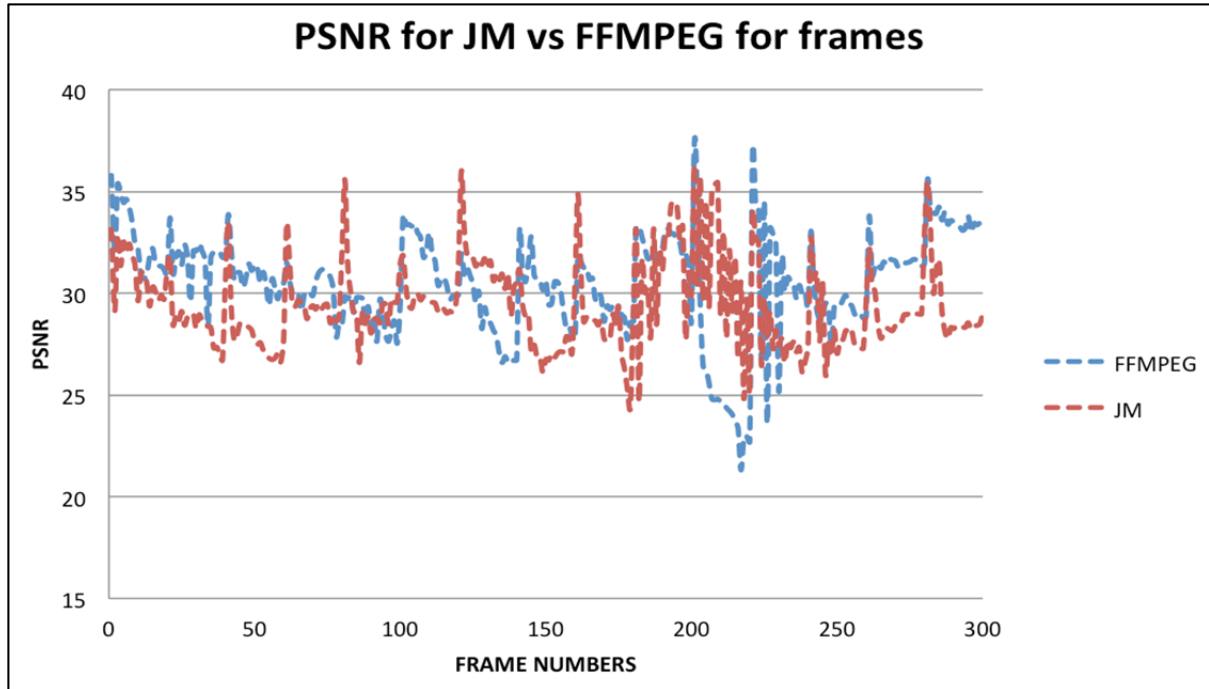


Figure 5.12. PSNR values of frames for JM vs FFMPEG with 5% slice loss for 1024 kbps Foreman sequence.

This proposed priority scheme is implemented on NAL unit, which contains one slice for each unit [29]. We first compute the CMSE (cumulative mean squared error) introduced in the GOP by the loss of one slice at time [30]. The slices of each GOP are then divided in four equally populated priority classes, such that 25% slices with highest GOP are assigned to highest priority class. Figure 5.13 describes the CMSE calculation process for FFMPEG.

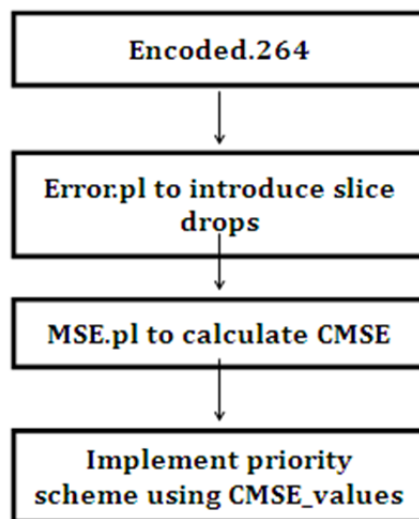


Figure 5.13. Description of the CMSE calculation process for FFMPEG.

Table 5.9 shows the percentage of I, P and B slices in all four priorities with different bit rates for Bus video sequence. These results also show the comparison between JM and FFMPEG codec. Similarly, Table 5.10 shows the percentage of I, P and B slices in all four priorities with different bit rates for Foreman video sequence as well as the comparison between JM and FFMPEG codec.

Table 5.9. Bus Showing Percentage I, P and B Slices in Different Priorities for Different Bit Rates in FFMPEG and JM

Bitrate (kbps)		256		512		1024	
Codec		JM	FFMPEG	JM	FFMPEG	JM	FFMPEG
% I frame slices	Priority 1	4.4	5	4.1	6.8	5.8	7.7
	Priority 2	6.8	9.3	5.2	8.2	7.4	7.5
	Priority 3	12	10.4	4.2	9	8.7	7.2
	Priority 4	15.1	9.6	4.8	9.9	8.7	7.7
% P frame slices	Priority 1	18.1	17.7	18.2	13.9	17.5	12.7
	Priority 2	12.4	9.3	15.6	11.7	15.1	13.2
	Priority 3	9.7	9.6	13.5	12.2	11.1	13.3
	Priority 4	7.1	13.6	13.6	14	6.4	16.5
% B frame slices	Priority 1	2.4	1.8	2.7	4.25	1.8	4.6
	Priority 2	5.7	6.8	4.1	5.1	2.5	4.4
	Priority 3	3.2	5	4.8	3.7	5.1	4.6
	Priority 4	2.8	1.9	6.6	1.1	9.9	0.85

Table 5.10. GOP 20 of Foreman Sequence Showing Percentage of I, P and B Slices in Different Priorities for Different Bit Rates in FFMPEG and JM

Bitrate (kbps)		512		1024	
Codec		JM	FFMPEG	JM	FFMPEG
% I frame slices	Priority 1	3.2	13.8	7.3	14
	Priority 2	3.25	10.3	6.2	8.4
	Priority 3	2.9	8.2	4.2	4.8
	Priority 4	2.4	5.7	1.5	4.3
% P frame slices	Priority 1	19.2	9.9	15.4	9.5
	Priority 2	18.5	12.3	15.8	13.2
	Priority 3	18.2	13.8	16.1	17.1
	Priority 4	18.3	17.1	17.9	19.6
% B frame slices	Priority 1	2.6	1.3	2.3	1.5
	Priority 2	3.3	2.4	3.05	3.4
	Priority 3	4	3	4.7	3
	Priority 4	4.3	2.3	5.6	1.2

CHAPTER 6

CONCLUSION AND FUTURE STUDY

The main aim of this thesis was to study and compare the two best open source codecs available for research on H.264 AVC video coding standard (i.e., JM and FFMPEG).

Our study involved the following:

- Study of FFMPEG codec.
- Analysis of all the features available in FFMPEG and x264 codec.
- Analysis of the objective and subjective video quality by considering PSNR and VQM measures, respectively.
- Observations based on PSNR values and rate control in FFMPEG versus JM in lossless and lossy environment.
- Observed different frames based on CMSE values to allot different priorities.

Following are our observations about their performance and features:

- FFMPEG showed better results in error-free or in lossy environment as compared to JM in test scenarios.
- The decoding and encoding speed in FFMPEG is almost 100 times faster than JM, making it suitable for real time encoding.
- There are many good rate control algorithms used in FFMPEG that can be used to further improve its performance.
- It can encode and send data on networks using RTP and also decode in real-time environments for streaming.
- However, FFMPEG does not support some of the H.264 features such as FMO (Flexible macroblock ordering) and Data Partitioning.
- Stability is also an issue with FFMPEG because its code is frequently changing in its improved versions.
- It does not have any concealment scheme for the whole frame loss.
- x264 in FFMpeg only supports Baseline, High and Main profile.

In future, the following studies may prove useful:

- The impact of various compression enhancing parameters on lossy channels to study the compression and error resiliency tradeoff.
- The study of various error resiliency schemes in FFMPEG.

REFERENCES

- [1] R. Schäfer, T. Wiegand, and H. Schwarz. "The Emerging H.264/AVC Standard." Last modified January 12, 2003. http://tech.ebu.ch/docs/techreview/trev_293-schaefer.pdf.
- [2] T. Connie, P. Nasiopoulos, V. C. M. Leung and Y. P. Fallah. "Video Packetization Techniques for Enhancing H.264 Video Transmission Over 3G Networks." In *Proc. IEEE Consumer Commun. Networking Conference*, 800-804. Piscataway: The Institute of Electrical and Electronics Engineers, 2008.
- [3] J. Hu, S. Choudary and J. D. Gibson. "H.264 Video Over 802.11a WLANs with Multipath Fading: Parameter Interplay and Delivered Quality." In *IEEE International Multimedia Conference and Expo*, 1790-1793. Beijing: The Institute of Electrical and Electronics Engineers, 2007.
- [4] T. Wiegand, G. J. Sullivan, G. Bjontegaard and A. Luthra. "Overview of the H.264/AVC Video Coding Standard." *IEEE Transactions on Circuits and Systems for Video Technology* 13, no. 7 (2002): 560-576.
- [5] Y. -C. Chu and M. -J. Chen. "High Performance Adaptive Deblocking Filter for H.264." *IEICE Transactions on Information and Systems* E89-D, no. 1 (2006): 367-371.
- [6] N. Ozbek and T. Tunali. "A Survey on the H.264/AVC Standard." Last modified November 15, 2005. <http://journals.tubitak.gov.tr/elektrik/issues/elk-05-13-3/elk-13-3-1-0408-5.pdf>
- [7] J. Ostermann, J. Bormans, P. List, D. Marpe, M. Narroschke, F. Pereira, T. Stockhammer, and T. Wedi. "Video Coding with H.264/AVC: Tools, Performance, and Complexity." Accessed June, 2010. <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1286980>
- [8] T. Stockhammer, M. Hannuksela, and T. Wiegand, "H.264/AVC in Wireless Environments." *IEEE Transactions on Circuits and Systems for Video Technology* 13, (2003): 657-673.
- [9] Merrit, L., and R. Vanam. "Improved Rate Control And Motion Estimation for H.264 Encoder." Paper presented at the IEEE International Conference on Image Processing, San Antonio, TX, September 2007.
- [10] Fraunhofer Heinrich Hertz Institute. "H.264/AVC Software Coordination." Last modified January 6, 2011. <http://iphome.hhi.de/suehring/tml/>.
- [11] FFMPEG. "FFMPEG Documentation." Last modified June 27, 2012. <http://ffmpeg.org/doxygen/trunk/index.html>.
- [12] VideoLAN. "x264." Last modified May 12, 2011. <http://developers.videolan.org/x264.html>.
- [13] Mewiki. "X264 Setting." Last modified March 19, 2012. http://mewiki.project357.com/wiki/X264_Settings#subme

- [14] Gonzalez, R. C., and R. E. Woods. *Digital Image Processing*. New Jersey: Prentice, 2007.
- [15] Sullivan, G. J., and T. Wiegand. "Rate-Distortion Optimization for Video Compression." *IEEE Signal Processing Magazine* 15, no. 6 (1998): 74-90.
- [16] FFMPEG. "FFmpeg." Last modified June 7, 2012. <http://ffmpeg.org/index.html>.
- [17] G. J. Sullivan, T. Wiegand, and K. P. Lim, "Joint Model Reference Encoding Methods and Decoding Concealment Methods." California, JVT-I046, 2003.
- [18] Google. "x264-FFmpeg Options Guide." Accessed August, 2011. <https://sites.google.com/site/linuxencoding/x264-ffmpeg-mapping>
- [19] X. Gao, C. J. Duanmu, and C. R. Zou. "A Multi-Level Successive Elimination Algorithm for Block Matching Motion Estimation." *IEEE Transactions on Image Processing* 9, no. 3 (2000): 501-504.
- [20] Compression. "Video Filtering and Compression by MSU Video Group." Last modified June 20, 2012. <http://www.compression.ru/video/index.htm>.
- [21] K. Seth, V. Kamakoti, and S. Srinivasan. "Motion Vector Recovery Based Error Concealment For H.264 Video Communication: A Review." *IETE Technical Review* 28, no. 1 (2011): 29-39.
- [22] S. Kumar, L. Xu, M. K. Mandal, and S. Panchanathan. "Error Resiliency Schemes in H.264/AVC Video Coding Standard." *Journal Visual Communication and Image Representation* 17, no. 2 (2006): 34-40.
- [23] UbuntuForums. "HOWTO: Install and Use the Latest FFmpeg and x264." Last modified June 20, 2012. <http://ubuntuforums.org/showthread.php?t=786095>.
- [24] M. Vranjes, S. Rimac-Drljie, D. Zagar. "Subjective and Objective Quality Evaluation of the H.264/AVC Coded Video." Paper presented at the International Conference on Systems, Signals and Image Processing, Bratislava, Slovakia, June 2008.
- [25] Bandyopadhyay, S. K., Z. Wu, P. Pandit, and J. M. Boyce. "An Error Concealment Scheme for Entire Frame Losses for H.264/AVC." Paper presented at the Proceedings of the IEEE Sarnoff Symposium, Princeton, NJ, March 2006.
- [26] M. Li, and B. Wang. "Whole Frame Loss Recovery Algorithm for H.264 Decoders." Accessed December, 2011. <http://www.computer.org/portal/web/csdl/doi/10.1109/IAS.2009>.
- [27] T. L. Lin, Y. L. Chang and P. C. Cosman. "Subjective Experiment and Modeling of Whole Frame Packet Loss Visibility for H.264." Paper presented at the International Packet Video Workshop, Hong Kong, China, December 2010.
- [28] N. Staelens, B. Vermeulen, S. Moens, J. F. Macq, P. Lambert, R. Van de Walle, and P. Demeester. "Assessing the Influence of Packet Loss and Frame Freezes on the Perceptual Quality of Full Length Movies." Paper presented at the International Workshop on Video Processing and Quality Metrics for Consumer Electronics, Scottsdale, AZ, 2009.

- [29] R. Pastrana Vidal, J. Gicquel, C. Colomes, and H. Cherifi. “Sporadic Frame Dropping Impact on Quality Perception.” *Proceedings of the SPIE Human Vision and Electronic Imaging* 5292, (2004):182–193.
- [30] S. Bandyopadhyay, Z. Wu, P. Pandit, and J. M. Boyce. “An Error Concealment Scheme for Entire Frame Losses for H.264/AVC.” Accessed January, 2010. <https://ipbt.hhi.fraunhofer.de>.

APPENDIX A

JM (H.264)

H.264 Video codec parameters (JM)

JM 14.4 Encoder and the JM 14.4 Decoder are used to do all the experiments. To allow it to handle errors more efficiently than the original, the decoder is modified for better performance.

Encoder.cfg Parameters:

Parameters we usually change in encoder.cfg. These are the values I have used in most of my simulations.

```

InputFile           = "foreman_cif.yuv" # Input sequence
InputHeaderLength   = 0                  # If the inputfile has a header, state it's length in
byte here
StartFrame          = 0                  # Start frame for encoding. (0-N)
FramesToBeEncoded   = 300               # Number of frames to be coded
FrameRate           = 30.0              # Frame Rate per second (0.1-100.0)
SourceWidth         = 352               # Source frame width
SourceHeight        = 288               # Source frame height
TraceFile           = "trace_enc.txt"   # Trace file
ReconFile           = "test_rec.yuv"    # Recontruction YUV file
OutputFile          = "NAL200.264"      # Bitstream
ProfileIDC          = 100               # Profile IDC
IntraProfile        = 0
LevelIDC            = 40 # Level IDC (e.g. 20 = level 2.0)

IntraPeriod         = 10 # Period of I-pictures (0=only first)
IDRPeriod           = 0 # Period of IDR pictures (0=only first)
NumberReferenceFrames = 6 # Number of previous frames used for inter motion search
(0-16)
RandomIntraMBRefresh = 0 # Forced intra MBs per picture – IMBR
NumberBFrames       = 1 # Number of B coded frames inserted (0=not used)
QPBSlice            = 30 # Quant. Param for B slices (0-51)

```

SymbolMode	= 0 # Symbol mode (Entropy coding method: 0=UVLC, 1=CABAC)
OutFileMode	= 0 # Output file mode, 0:Annex B, 1:RTP
PartitionMode	= 0 # Partition Mode, 0: no DP, 1: 3 Partitions per Slice
SliceMode	= 2 # Slice mode (0=off 1=fixed #mb in slice 2=fixed #bytes in slice 3=use callback)
SliceArgument	= 150 # Slice argument (Arguments to modes 1 and 2 above)
num_slice_groups_minus1	= 1 # Number of Slice Groups Minus 1, 0 == no FMO, 1 == two slice groups, etc.
RestrictSearchRange	= 2 # restriction for (0: blocks and ref, 1: ref, 2: no restrictions)
RDOptimization	= 1 # rd-optimized mode decision
UseConstrainedIntraPred	= 1 # If 1, Inter pixels are not used for Intra macroblock prediction, CIP
RateControlEnable	= 1 # 0 Disable, 1 Enable
Bitrate	= 512000 # Bitrate(bps) varies depending on requirement

APPENDIX B

FFMPEG (H.264)

Installation Of FFMPEG on Ubuntu- 11.04

Description for installation of FFMPEG latest git-master version on Ubuntu 11.04. We installed FFMPEG git-master version “master-git-N-30860-g83f9bc8” for all decoding and the x264 encoder version “r2008m 4c552d8” for all encoding on Ubuntu 11.04 and the steps to install FFMPEG and x264 on different Ubuntu versions other than the above can vary.

Install the dependencies by following commands on terminal-

- 1) Uninstall x264, libx264-dev, and FFMPEG if they are already installed. Open a terminal and run the following code-

code-

```
[sudo apt-get remove FFMPEG x264 libx264-dev]
```

- 2) To get the dependencies to install FFMPEG and x264.

code-

```
[sudo apt-get update
```

```
sudo apt-get install build-essential checkinstall git libfaac-dev libjack-jackd2-dev \
libmp3lame-dev libopencore-amrnb-dev libopencore-amrwb-dev libssl1.2-dev libtheora-
dev libva-dev \libvdpau-dev libvorbis-dev libx11-dev libxfixes-dev texi2html yasm
zlib1g-dev]
```

- 3) Get source file, compile and install x264.

code-

```
[cd
```

```
git clone git://git.videolan.org/x264
```

```
cd x264
```

```
./configure --enable-static
```

```
make
```

```
sudo checkinstall --pkgname=x264 --pkgversion="3:$(./version.sh | \
awk -F[" ] ' /POINT/{print $4"+git"$5}' )" --backup=no --deldoc=yes \
--fstrans=no --default]
```

- 4) Get source file, compile and install FFMPEG.

code-

```
[cd
git clone --depth 1 git://source.FFMPEG.org/FFMPEG
cd FFMPEG
./configure --enable-gpl --enable-libfaac --enable-libmp3lame --enable-libopencore-
amrnb \
--enable-libopencore-amrwb --enable-libtheora --enable-libvorbis --enable-libx264 \
--enable-nonfree --enable-postproc --enable-version3 --enable-x11grab
make
sudo checkinstall --pkgname=FFMPEG --pkgversion="5:$(date +%Y%m%d%H%M)-
git" -
-backup=no \
--deldoc=yes --fstrans=no --default
hash x264 FFMPEG ffplay ffprobe]
```

- 5) To allow x264 to accept command from FFMPEG or access directly x264.

code-

```
[cd ~/x264
make distclean
./configure --enable-static
make
sudo checkinstall --pkgname=x264 --pkgversion="3:$(./version.sh | \
awk -F[" ' ] ' /POINT/{print $4"+git"$5}')" --backup=no --deldoc=yes \
--fstrans=no --default]
```

FFMPEG and x264 command line:

- 1) 1-pass encoding in FFMPEG-

```
FFMPEG -i input -acodec libfaac -aq 100 -b:v 256k -r 30 -s 352x288 -vcodec
libx264 -preset medium -profile baseline -level 3.0 output.mp4
```

- 2) 2-pass encoding in FFMPEG-

First pass-

```
FFMPEG -i input -pass 1 -acodec libfaac -aq 100 -b:v 256k -r 30 -s 352x288 -
vcodec libx264 -preset medium -profile baseline -level 3.0 /dev/null
```

Second pass-

```
FFMPEG -i input -pass 2 -acodec libfaac -aq 100 -b:v 256k -r 30 -s 352x288 -
vcodec libx264 -preset medium -profile baseline -level 3.0 -y out.264
```

- 3) To use x264 directly from FFMPEG command-line-

```
FFMPEG -i input -acodec libfaac -aq 100 -b:v 256k -r 30 -s 352x288 -vcodec
libx264 -preset medium -x264opts slice-max-size=150:bframes=1:keyint=20 -
profile baseline -level 3.0 output.mp4
```

- 4) Decoding in FFMPEG-

```
FFMPEG -r 30 -i input.264 -r 30 -y out.yuv
```

APPENDIX C

PROGRAM MODULES

h264_analyze (executable)

This file is used to obtain the NAL unit information and the trace file from an h264 encoded file. It provides the slice type, memory address and other important information for every slice. It will give two text files including one with only NAL unit information and another file with everything.

Usage: ./h264_analyze <input.264> > tracefile.txt

hardik_error (executable)

This file is to insert error or in other words, remove the particular slices as per the requirement.

Usage: ./hardik_error -v \$starting_memory_address:\$size_to_remove input.264 > out.264

random_err-insert.pl

This file is used to insert random errors in encoded .264 files. The user has to give the slices that they want removed in a text file and .264 file as input to this script.

Usage: perl random_err-insert.pl

error.pl

This script is used in the calculation of IMSE values. This script uses hardik_error to drop each and every slice one by one and decodes it at the same time to calculate MSE values.

Input for this script is .264 file, NAL parser file and it also uses FFMPEG to finally decode the

.264 files.

Usage: perl error.pl

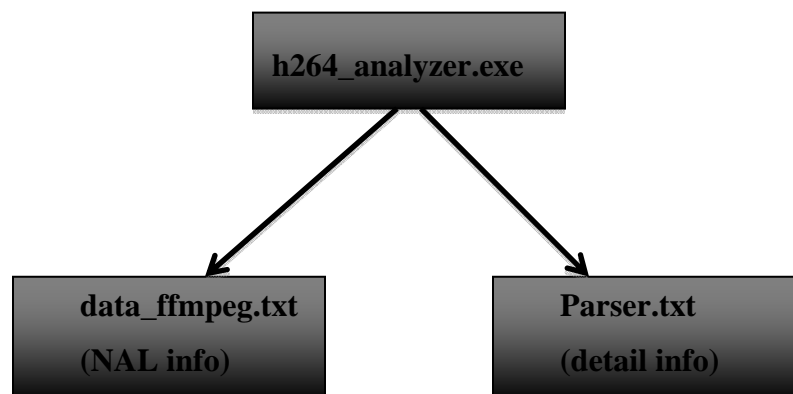
MSE_calc.pl

This script uses MSE matlab executable to calculate IMSE and then CMSE and puts all the information in a text file.

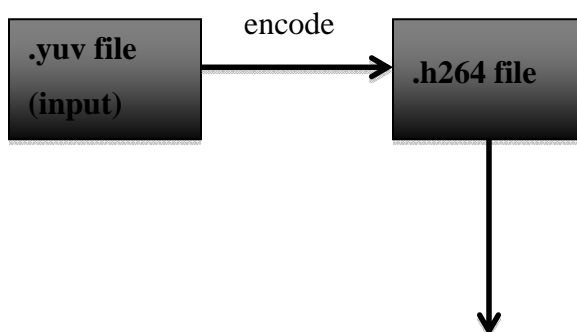
Usage: perl MSE_calc.pl

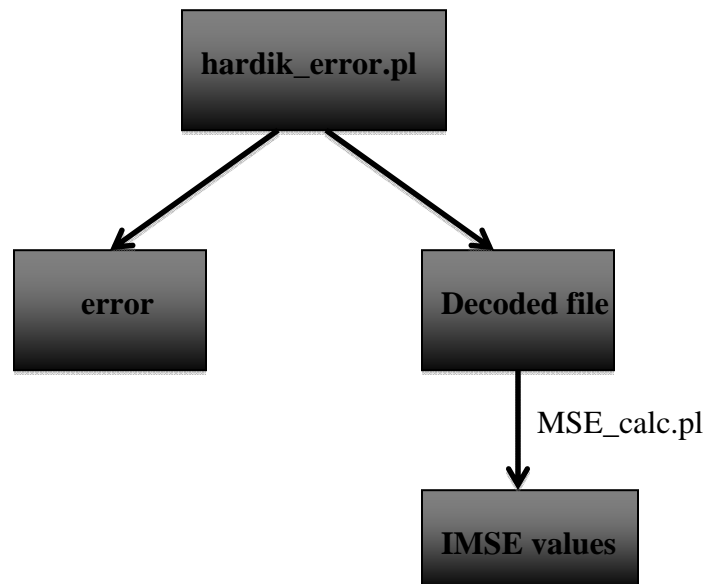
Steps to follow during experiment -

Parsing steps:



To calculate IMSE:





For random error drop:

