

University of Southern Queensland
Faculty of Engineering & Surveying

Video Compression using ITU-T Recommendation H.264

A dissertation submitted by

B. Farmer

in fulfilment of the requirements of

ENG4112 Research Project

towards the degree of

Bachelor of Engineering(Computer Systems)

Submitted: October, 2005

Abstract

The H.264 standard defines a compliant bitstream and methods of decoding this bitstream to reconstruct a video sequence.

This research project introduces H.264 video compression techniques including FMO, CAVLC, Deblocking Filter, logarithmic quantisation and integer transformation.

The encoding process for the baseline profile is discussed in detail, specific implementation procedures detailed and references made to available source code. Encoders are required to provide conforming bitstreams consistent with their stated profile and level.

A decoder that declares its conformance to a specific profile must be able to support all of the features of that profile. Interoperability between individual encoders and decoders is achieved through this profile and level compliance.

The provision of a GUI allows user interaction with the encoder and decoder. Parameters may be varied in order to suit particular applications. The variation of parameters require tradeoffs between fidelity, file size and bit rate to be made.

University of Southern Queensland
Faculty of Engineering and Surveying

ENG4111/2 <i>Research Project</i>
--

Limitations of Use

The Council of the University of Southern Queensland, its Faculty of Engineering and Surveying, and the staff of the University of Southern Queensland, do not accept any responsibility for the truth, accuracy or completeness of material contained within or associated with this dissertation.

Persons using all or any part of this material do so at their own risk, and not at the risk of the Council of the University of Southern Queensland, its Faculty of Engineering and Surveying or the staff of the University of Southern Queensland.

This dissertation reports an educational exercise and has no purpose or validity beyond this exercise. The sole purpose of the course pair entitled “Research Project” is to contribute to the overall education within the student’s chosen degree program. This document, the associated hardware, software, drawings, and other material set out in the associated appendices should not be used for any other purpose: if they are so used, it is entirely at the risk of the user.

Prof G Baker

Dean

Faculty of Engineering and Surveying

Certification of Dissertation

I certify that the ideas, designs and experimental work, results, analyses and conclusions set out in this dissertation are entirely my own effort, except where otherwise indicated and acknowledged.

I further certify that the work is original and has not been previously submitted for assessment in any other course or institution, except where specifically stated.

B. FARMER

0011120187

Signature

Date

Acknowledgments

I would like to thank my supervisor, Dr Wei Xiang for his assistance and guidance throughout this research project.

On a personal note I would thank my husband, Aaron for his support not only through this research project but throughout my entire degree.

Finally to my daughter, Jordi who arrived half way through this year and my son, Samuel, both of whom are a welcome distraction from study, I thank you both for your patience and love.

B. FARMER

University of Southern Queensland

October 2005

Contents

Abstract	i
Acknowledgments	iv
List of Figures	xi
List of Tables	xiv
Glossary	xvi
Chapter 1 Introduction	1
1.1 Reference Software	2
1.2 Video Compression	2
1.3 Overview of the Dissertation	3
Chapter 2 Video Compression Techniques	5
2.1 Chapter Overview	5
2.2 Video Compression Encoding	6

CONTENTS	vi
2.3 Video Compression Decoding	6
2.4 Frame Components	7
2.5 Video Compression Techniques	7
2.5.1 Motion Compensation	8
2.5.2 Spatial Prediction	9
2.5.3 Transformation	10
2.5.4 Quantisation	10
2.5.5 Entropy Coding	11
2.6 Evolution of MPEG and H26L Codecs	11
2.7 Chapter Summary	13
Chapter 3 The H.264 Recommendation	14
3.1 Chapter Overview	14
3.2 H.264 Profiles	15
3.3 Bitstream Formats	17
3.4 Network Abstraction Layer	18
3.4.1 Data Partitioning	19
3.5 Slices	19
3.6 Motion Compensated Prediction	20
3.7 Flexible Macroblock Ordering	21
3.8 Arbitrary Slice Ordering	22

3.9 Integer Transform	22
3.10 Logarithmic Quantisation	23
3.11 Entropy Encoding	23
3.12 Adaptive Deblocking Filter	24
3.13 Video Formats	24
3.14 Hypothetical Reference Decoder	25
3.15 Chapter Summary	26
Chapter 4 H.264 Encoder Principles	27
4.1 Chapter Overview	27
4.2 The Encoder	28
4.3 The Encoded Bitstream	29
4.3.1 Parameter Sets	31
4.3.2 Intra Video Coding Layer Network Access Layer Unit	32
4.3.3 Inter Video Coding Layer Network Access Layer Unit	48
4.4 Chapter Summary	50
Chapter 5 H.264 Decoding Principles	52
5.1 Chapter Overview	52
5.2 The Decoder	53
5.3 The Bitstream	54

5.3.1	Parameter Sets	54
5.3.2	Intra Video Coding Layer Network Access Layer Unit	55
5.4	Chapter Summary	62
Chapter 6 Graphical User Interface		63
6.1	Chapter Overview	63
6.2	Main Interface	63
6.3	Encoder Interface	64
6.4	Decoder Interface	76
6.5	Chapter Summary	77
Chapter 7 Results		78
7.1	Chapter Overview	78
7.2	Parameter Effects	79
7.3	Conforming Bitstream and Fidelity Testing	80
7.3.1	Methodology	81
7.3.2	Results	81
7.4	Fidelity and Storage Comparison	82
7.5	Peak Signal to Noise Ratio	83
7.6	User Software Interaction	85
7.7	Chapter Summary	85

Chapter 8 Conclusion **86**

8.1 Achievement of Objectives 86

8.2 Future Work 87

List of References **88**

Appendix A Project Specification **91**

Appendix B NAL and Picture Parameters **93**

B.1 Chapter Overview 93

B.2 Network Access Layer Parameters 93

B.3 Slice Parameters 95

B.4 Macroblock Parameters 98

B.5 CAVLC Parameters 99

Appendix C Sequence Parameter Set **101**

C.1 Chapter Overview 101

C.2 Parameters 101

C.3 Bitstream information 104

C.4 SPS NALU Bitstream 108

Appendix D Picture Parameter Set Bitstream **109**

D.1 Chapter Overview 109

D.2 Parameters	109
D.3 Bitstream info	113
D.4 PPS NALU Bitstream	116
Appendix E VCL Bitstream	117
E.1 Chapter Overview	117
E.1.1 Slice Header Bitstream	117
E.1.2 Macroblock Header Bitstream	118
E.1.3 Macroblock Image Bitstream	119
E.2 Intra VCL NALU Bitstream	126

List of Figures

3.1	Slice group maps available for use with Flexible Macroblock Ordering.	22
4.1	H.264 Encoder Block Diagram	28
4.2	The original first frame from the container_qcif.yuv video sequence . . .	33
4.3	A magnified 4×4 block of macroblocks of the top left corner of the first frame of the original YUV video.	36
4.4	Integer Transformation Method.	39
4.5	Second frame of container_qcif.yuv sourced from Kansas State University.	48
4.6	Magnified view of 2×2 block of macroblocks from the first frame of the original YUV video.	49
4.7	Magnified view of 2×2 block of macroblocks from the second frame of original YUV video.	49
5.1	H.264 Decoder Block Diagram	53
5.2	The decoded frame of container_qcif.	53
5.3	The second frame of container_qcif.264 decoded without the deblocking filter.	61

6.1	Start Screen of H264 Baseline Software	64
6.2	Information Screen of H264 Baseline Encode.	64
6.3	Main Screen of H264 Baseline Encode.	65
6.4	Picture Screen of H264 Baseline Encode.	66
6.5	Control Tab of H264 Baseline Encode.	67
6.6	FMO Tab of H264 Baseline Encode - No FMO.	69
6.7	FMO Tab of H264 Baseline Encode - Interleave Slice Map.	70
6.8	FMO Tab of H264 Baseline Encode - Dispersed Slice Map.	71
6.9	FMO Tab of H264 Baseline Encode - Foreground with left-over Slice Map.	71
6.10	FMO Tab of H264 Baseline Encode - Box-out Slice Map.	72
6.11	FMO Tab of H264 Baseline Encode - Raster Scan Slice Map.	72
6.12	FMO Tab of H264 Baseline Encode - Wipe Slice Map.	73
6.13	FMO Tab of H264 Baseline Encode - Explicit Slice Map.	73
6.14	Filter Tab of H264 Baseline Encode.	74
6.15	Rate Control Tab of H264 Baseline Encode.	74
6.16	Misc Tab of H264 Baseline Encode.	75
6.17	Main Screen of H264 Baseline Decode.	76
6.18	Advanced Screen of H264 Baseline Decode.	77
7.1	a. Decoded Frame using WinDVD Platinum. b. Decoded Frame using ImToo MPEG Encoder.	81

7.2 a. MPEG 1 b. MPEG 2 c. DVD Format. 83

7.3 A decoded P Slice with an average PSNR values of 38.33dB. 84

List of Tables

4.1	Differences between original u chroma values and best mode predicted values.	36
4.2	The first 4×4 block of luma values of the raw container_qcif.yuv video. .	37
4.3	Calculated luma differences of the first 4×4 block.	37
4.4	The Integer Transform matrix used for 4×4 blocks.	38
4.5	The 4×4 block following horizontal transform.	39
4.6	The 4×4 block of transformed coefficients.	40
4.7	The 4×4 block of quantised values.	42
4.8	4×4 block of dequantised values.	43
4.9	4×4 block of inverse transformed values.	43
4.10	The first 4×4 luma block of macroblock 35 from frame 1.	50
4.11	The first 4×4 luma block of macroblock 35 from original frame 2.	50
5.1	The first 4×4 luma block following entropy decoding and rescaling. . . .	59
5.2	The first 4×4 decoded luma block following horizontal inverse transform.	60

5.3	The first 4×4 decoded luma block following the inverse vertical transform.	61
7.1	Effects of Parameter Changes.	80

Glossary

<i>ABT</i>	<i>ABT</i> Adaptive Block-size Transform
<i>AVC</i>	<i>AVC</i> Advanced Video Coding
<i>ASO</i>	<i>ASO</i> -Arbitrary Slice Ordering
<i>CABAC</i>	<i>CABAC</i> - Context Adaptive Binary Arithmetic Coding
<i>CAVLC</i>	<i>CAVLC</i> - Context Adaptive Variable Length Coding
<i>CBP</i>	<i>CBP</i> Coded Block Pattern
<i>Chroma</i>	<i>Chroma</i> -Chrominance or Colour
<i>CIF</i>	<i>CIF</i> -Common Intermediate Format
<i>CODEC</i>	<i>CODEC</i> -Encoder/Decoder pair
<i>DCT</i>	<i>DCT</i> -Discrete Cosine Transform
<i>DPB</i>	<i>DPB</i> -Decoded Picture Buffer
<i>DVD</i>	<i>DVD</i> -Digital Versatile Disk
<i>EBSP</i>	<i>EBSP</i> - Emulation Byte Sequence Payload
<i>FME</i>	<i>FME</i> - Fast Motion Estimation
<i>FMO</i>	<i>FMO</i> -Flexible Macroblock Ordering
<i>GOB</i>	<i>GOB</i> -Group of Blocks
<i>GOP</i>	<i>GOP</i> -Group of Pictures
<i>GUI</i>	<i>GUI</i> -Graphical User Interface
<i>HRD</i>	<i>HRD</i> -Hypothetical Reference Decoder
<i>IDR</i>	<i>IDR</i> -Instantaneous Decoding Refresh

<i>IEC</i>	<i>IEC</i> -International Electrotechnical Commission
<i>IGOP</i>	<i>IGOP</i> -Image Group of Pictures
<i>ITU</i>	<i>ITU</i> -International Telecommunications Union
<i>ISO</i>	<i>ISO</i> -International Standards Organisation
<i>LSB</i>	<i>LSB</i> -least significant bit
<i>Luma</i>	<i>Luma</i> -Luminance or Brightness
<i>MPEG</i>	<i>MPEG</i> -Motion Pictures Expert Group
<i>MSB</i>	<i>MSB</i> -most significant bit
<i>MSE</i>	<i>MSE</i> -Mean Squared Error
<i>NAL</i>	<i>NAL</i> -Network Abstraction Layer
<i>NALU</i>	<i>NALU</i> -Network Abstraction Layer Unit
<i>NNZ</i>	<i>NNZ</i> -Number of Nonzero Coefficients
<i>PEL</i>	<i>PEL</i> -Picture Element
<i>POC</i>	<i>POC</i> -Picture Order Count
<i>PPS</i>	<i>PPS</i> -Picture Parameter Set
<i>PSNR</i>	<i>PSNR</i> - Peak Signal to Noise Ratio
<i>QCIF</i>	<i>QCIF</i> -Quarter Common Intermediate Format
<i>QP</i>	<i>QP</i> -Quantisation Parameter
<i>RBSP</i>	<i>RBSP</i> -Raw Byte Sequence Payload
<i>SAD</i>	<i>SAD</i> -Sum of Absolute Differences
<i>SATD</i>	<i>SATD</i> -Sum of Absolute Transform Differences
<i>SEI</i>	<i>SEI</i> -Supplemental Enhancement Information
<i>SODB</i>	<i>SODB</i> -String of Data Bits
<i>SPS</i>	<i>SPS</i> -Sequence Parameter Set
<i>VCEG</i>	<i>VCEG</i> -Video Coding Experts Group
<i>VCL</i>	<i>VCL</i> - Video Coding Layer
<i>VLC</i>	<i>VLC</i> - Variable Length Coding

Chapter 1

Introduction

The aim of this research project is to investigate the International Telecommunications Union - Telecommunications (ITU-T) Recommendation H.264 and implement the video compression algorithms.

ITU-T Recommendation H.264 is also known as International Standards Organisation / International Electrotechnical Commission 14496 Part 10 or MPEG-4 Advanced Video Coding becoming a standard in May 2003.

The ITU-T and the ISO / IEC are the two independent global standardisation organisations for telecommunications.

The groups that are specifically responsible for video compression within these organisations is the Joint Technical Committee (JTC), Sub Committee 29, Working Group 11 (WG11). The ISO/IEC JTC1/SC29/WG11, is more commonly known as Moving Picture Experts Group (MPEG). The Video Coding Experts Group (VCEG) is the ITU-T video compression group.

H.264 development was first initiated by the VCEG and became a collaborative effort of both the VCEG and MPEG when a Joint Video Team (JVT) was established in December 2001.

The H.264 Recommendation sought to respond to demands for better compression of

video and global interoperability. The Recommendation may be applied to many applications, including videoconferencing, television broadcasting, data storage, internet streaming and communication.

The H.264 standard defines the parts of a compliant bitstream as well as the decoder processing methods required to turn the bitstream back into video. The standard intends to allow commercial applications to develop their own individuality, whilst maintaining the global interoperability.

1.1 Reference Software

This research project used Joint Video Team (2004) Joint Model 9.0 (JM90) reference software, that was last modified in October 2004. The reference software is constantly being updated and the most recent software is called JM10.1. The reference software was developed in conjunction with the Recommendation to test the viability of various compression techniques.

The reference software has been refined during the course of this research project in order provide a user friendly interface to the baseline profile of the H.264 Recommendation. These changes maintain a conforming bitstream and utilise the decoder processing methods as defined by the Recommendation, whilst seeking to optimise the code with respect to encoding and decoding complexity.

The software is written in C/C++ and is evaluated against other commonly used codecs for fidelity, complexity and encoded data storage size.

1.2 Video Compression

Video compression provides the ability to encode and decode a stream of consecutive digital images allowing for efficient transmission and storage. This is achieved using a variety of video compression algorithms that compact or condense a digital video sequence into a smaller number of bits than of original raw video.

A digital video sequence samples a natural time varying real world event at consistent time intervals converting the image to data. Frame samples are commonly taken at 25 or 30 frames per second to reproduce television quality moving video. Video compression codecs are characterised by differing tradeoffs, such as complexity, encoded image data size and the reproduced fidelity.

Video compression research started in the mid-1960s, whereby each picture within the video stream was compressed separately and without reference to other pictures. The separate compression of pictures is used in the JPEG standard and the video formed by multiple consecutive pictures is known as motion JPEG.

1.3 Overview of the Dissertation

Chapter 2 discusses the typical video compression encoding and decoding methods, from raw video through to the encoded data being ready for storage or transmission. Common video compression techniques include motion compensation, transformation, quantisation and entropy coding. This chapter further discusses the evolution of the MPEG and H.26L series of codecs from the release of H.261 in 1990.

The three original profiles of H.264 are defined in Chapter 3. This chapter also discusses video compression techniques particular to the H.264 standard including flexible macroblock ordering (FMO), arbitrary slice order (ASO), adaptive deblocking filter, context adaptive variable length coding (CAVLC), improved motion compensation methods and logarithmic quantisation.

Chapter 4 provides a detailed insight into the algorithms used to compress the video image. This chapter follows a 4×4 frame block through the transformation, prediction, quantisation and entropy encoding process.

Chapter 5 follows the H.264 Recommendation for decoding a conforming bitstream. This chapter seeks to provide the mathematical approach to entropy decoding, inverse quantisation and inverse transform.

Chapter 6 discusses the Graphical User Interface (GUI) allowing the user to investigate various video compression parameters. The careful use of these parameters should allow the user to determine a fidelity and data size combination. The GUI provides seven tabs of encoder parameters that may be defined by the user.

Chapter 7 is the results chapter which discusses the fidelity, complexity, bit rate and file size of the H.264 Recommendation. The H.264 software is evaluated with respect to a variety of internal parameters, against other codecs, ease of use, signal to noise ratio and for bitstream conformance.

Chapter 8 concludes the dissertation with a discussion on the achievement of the objectives of the research project. Future work relating to H.264 Recommendation and for video compression is identified.

This dissertation is organised as follows:

Chapter 2 discusses common video compression techniques;

Chapter 3 identifies techniques particular to the H.264 standard, including FMO, ASO, CAVLC;

Chapter 4 follows encoding algorithms from raw video to an encoded video sequence;

Chapter 5 details a decoding method for decoding a conforming H.264 bitstream;

Chapter 6 the GUI allows for user interaction with a variety of video compression techniques;

Chapter 7 extensively evaluates the software for variations in fidelity, file size and complexity;

Chapter 8 concludes the dissertation and suggests further work in the area of the H.264 Recommendation.

Chapter 2

Video Compression Techniques

2.1 Chapter Overview

A digital image seeks to replicate a natural scene with respect to the colour, shape, brightness and texture of the real world. Digital images taken at regular intervals and displayed consecutively produce motion video.

This chapter seeks to provide an insight into the typical video compression procedures for encoding and decoding consecutive digital images.

Each digital image is divided into smaller component parts known as slices and macroblocks. Smaller sections of the image allow for more accurate video compression to be achieved and the composition of these blocks are discussed further in the chapter.

Common video compression techniques described in this chapter include motion compensation prediction methods, transformation, quantisation and entropy coding.

Finally a brief discussion on the evolution of video compression for both MPEG and the H.26L series of codecs is included.

2.2 Video Compression Encoding

Typical video encoding occurs as follows:

An encoder takes in an incoming video bitstream and codes the video, picture by picture, resulting in an encoded video sequence. This sequence should contain a smaller number of bits than the original video whilst minimising the encoder induced distortion.

Each picture is predicted with respect to time, whereby one or more previous or future frames may be referenced and the current frame predicted. The temporal differences between the predicted and the real frame is referred to as motion compensated prediction and produces a residual frame. The motion compensated prediction is typically described by a set of motion vectors.

The similarities between neighbouring pixel blocks following temporal prediction of the residual frame are exploited to produce residual samples. Spatial redundancy may allow the data size of the picture to be reduced.

Following the spatial prediction the residual samples are transformed to become transform coefficients. The transform coefficients then undergo quantisation in order to allow the coefficients to be represented by a designated number of bits.

The quantised coefficients along with the motion vectors will be compressed by an entropy encoder to produce a compressed bitstream. An entropy encoder exchanges short binary codes for commonly occurring vector or coefficient sequences.

2.3 Video Compression Decoding

Typically video decoding is conducted in the reverse order. The incoming bitstream is entropy decoded and inverse transformed to recreate a version of the original image.

The resulting decoded video sequence will be distorted when compared to the original, meaning a reduction in fidelity. This distortion directly relates to the redundancy techniques, quantisation and transformation.

2.4 Frame Components

Each individual picture within a video sequence is referred to as a frame and represents a snapshot image at a particular instant in time. Each frame is further divided into smaller components being slices and macroblocks.

Slices consist of consecutive macroblocks and they represent regions of a given frame. A frame can contain one or more slices and each slice will employ identical video compression techniques. Slices are able to be decoded independently of other slices.

A macroblock is a group of 16×16 picture elements and consists of luma and chroma picture information. Video compression standards seek to condense the macroblock data through the use of a number of video compression algorithms. A macroblock may be further divided into smaller block sizes depending on the compression techniques employed and the processing power available to the codec.

2.5 Video Compression Techniques

The two types of video compression techniques are lossless and lossy compression. Lossless compression reproduces the original video perfectly and lossy compression produces a distorted representation of the original.

Lossless compression is achieved by removing statistical redundancy from the original video stream. Statistical redundancy is the replacing of commonly occurring image bit patterns and replacing them with a reduced number of bits to represent the common bit pattern.

Lossy compression attempts to compress the size of the video sequence by removing elements that may negatively affect the fidelity of the encoded video sequence.

2.5.1 Motion Compensation

Motion compensation is using one or more video frames to predict a future frame. A television quality video sequence will typically use a sampling rate of 25 to 30 frames per second. A high sampling rate means the motion difference between consecutive frames is typically small.

The most simple prediction method for motion compensation is to produce a residual frame from the differences between a past frame with the current frame. The residual frame created contains the motion information from this portion of the video sequence.

Block Based Motion Compensation

A macroblock is a 16×16 pixel region of a frame that is compared with macroblocks from previously encoded frames to find a visually matching macroblock. Motion estimation is the name of the process for finding the macroblock with the best match. This chosen region is used as the prediction sample from which the difference between the current and the predicted macroblock is found. A residual block is created from the difference. This process is known as motion compensation. A motion vector transmits the residual information to provide positional information for the decoder to locate the prediction sample.

The main disadvantage in using block based motion estimation and compensation is due to the fact that real world objects are rarely square.

Improving Block Based Motion Compensation

Better motion compensation may be produced by the use of smaller block sizes than that of a complete macroblock. Using smaller block sizes will increase the complexity of the encoded video sequence requiring an increased number of searches to find the best possible match. Reduced residual information of smaller blocks results in an increase in the number of motion vectors required.

H.264 allows for improved motion compensation through adaptive block sizes of 4×4 , 8×8 and 16×16 .

Motion compensation can also be improved using interpolation. This compensation method starts from a block identified as the best match, but not identical to the block to be predicted. This best match block will now be interpolated with each of its neighbouring blocks to produce a new midway block, or half sample block, to be checked for a match. Once the best match has been achieved a new midway block, or quarter sample block, will be interpolated to derive the best prediction possible. This method of motion compensation is known as sub-pixel prediction. As the focus of the motion estimation is further defined the value provided from each subsequent improvement in the motion compensated residual frame is less than the gain from the previous interpolation.

Why not Motion Compensation?

Intra prediction is the encoding of an image without motion compensation. Inter prediction is the encoding of an image using motion compensation.

Motion compensation may not always produce the best compression of a video frame, particularly when the current frame differs significantly from previous frames of the video sequence. In this case an intra predicted frame may be required.

Each picture is predicted with respect to time. One or more previous or future frames are referenced and the current frame is predicted. The temporal differences between the predicted and the real frame is referred to as motion compensated prediction producing a residual frame. This prediction is described by a set of positional motion vectors.

2.5.2 Spatial Prediction

Spatial prediction is also known as intra prediction and refers to using previously encoded samples from the current frame for the prediction of the current sample. Sullivan and Wiegand (2005) define intra coding to be where ‘the picture is coded without re-

ferring to other pictures in a video sequence.’ This was one of the earliest developed prediction methods used in video compression algorithms.

An advantage of regularly only using intra prediction for complete frames is to allow for error resilience. This provides entry points into the bitstream which enables random access by a decoder ensuring any corrupted data is not repeated beyond this point. A complete picture coded using intra coding is referred to as an instantaneous decoding (IDR) refresh picture.

2.5.3 Transformation

Video compression transformation seeks to transform the motion compensated residual frame into another domain. The transform should be reversible to allow it to be decoded.

By transforming a complete 4×4 block of image pixel values, a transform block of 16 coefficient values is derived. No saving in bit information is made if 16 values are to be transmitted. Transformation allows a few coefficients to be transmitted to reproduce a distorted representation of the original 4×4 block.

Humans are sensitive to lower frequencies. Transforming the residual frame into frequency domain coefficients and passing them through a low pass filter will remove higher frequency content. A reduction in detail still allows the image to be recognisable.

2.5.4 Quantisation

Quantisation effectively smoothes the visual image by inducing distortion into the encoded image reducing fidelity. Quantisation is a lossy compression technique where detail removed from the image cannot be replaced.

The distortion induced in the coded image is a result of the compression achieved by quantization. The quantisation parameter is an integer amount that specifically details the precision allowed of transformed values. The minimum amount of precision

is referred to as the step size.

2.5.5 Entropy Coding

Entropy coding is the process used to replace commonly occurring bit patterns derived from vectors or motion prediction coefficients with short binary strings. Entropy coding is a lossless video compression technique that further compresses the output of the image data.

There are many types of entropy coding available. These have varying degrees of success and complexity. Huffman coding is a type of entropy coding scheme that assumes that certain bit strings will occur commonly throughout the image. The condensed bit strings produced by Huffman entropy coding are fixed and cannot be changed if inappropriate. Context adaptive entropy coding allows the short binary strings to change depending on the requirements of the bitstream.

This video compression stage is typically the last stage in the image encoding process prior to transmission or storage of the encoded data.

2.6 Evolution of MPEG and H26L Codecs

A codec is the name given to an encoder and decoder pair. The success of video compression techniques and their inclusion in a codec is a result of many factors. Delays in the commercial implementation of video compression techniques may be partially attributed to processing power requirements.

Modern video compression methods date back to the first version of ITU-T Recommendation H.120 released in 1984. H.120 improved compression efficiency by employing a temporal redundancy method known as conditional replenishment. Data is passed to indicate which areas of the picture can be repeated and which area will have its own separate compression information. This method simply replaces parts of the image that have been changed with new encoded information.

The second version of H.120 released in 1988 introduced spatial prediction techniques. A linear quantisation method was used in all of the video compression techniques prior to the H.264 standard.

H.261 was first released in 1990 and was the earliest hybrid codec standardised. Hybrid codecs used both motion prediction and transformation. All H26L series and MPEG standards from 1990 are hybrid standards.

ISO Standard MPEG-1 was released in 1993 and provided improved motion vector support. This standard was the first to allow half sample interpolation for motion estimation as well as bipredictive prediction. The motion compensated algorithms included in the MPEG-1 standard were complex and stretched the processing power that was available at that time. A two dimensional discrete cosine transform (DCT) of size 8×8 was used in both the MPEG-1 and MPEG-2 standards.

H.262 is also known as MPEG-2 and was released in 1994. This was the first joint standard to be developed and agreed between the ITU and ISO.

H.263 was released in 1995 and was primarily developed for videoconferencing and streaming applications. The second version of this H.263 standard first allowed the use of multiple reference frames for motion compensated prediction and the concept of B slices was introduced. H.263 and MPEG-4 Part 2 introduced the concept of varying block sizes and allow block sizes of 16×16 and 8×8 .

The MPEG-4 Part 2 Visual standard released in 1999 is a highly flexible standard including many different profiles. These profiles were each optimised for individual applications, such as data storage or internet streaming.

H.264 also known as MPEG-4 Advanced Video Codec (AVC) development began in 2001 and was formally released in May 2003. This new standard sought to achieve efficiency and reliability in its algorithm design. Three profiles were initially defined as part of the original standard reducing the number defined in the MPEG-4 Part 2 standard.

H.264 increases the concept of varying block sizes, by including the additional sizes of

8×16 , 16×8 , 8×4 , 4×8 and 4×4 to those allowed by H.263. H.264 uses a logarithmic quantisation method whereby a change in 6 results in a doubling of the quantisation step size.

H.264 introduces an adaptive deblocking filter and replaces the previous transformation method with an integer transform. The context adaptive entropy coding methods used by H.264 allow the compression of the bitstream to no longer assume which bit patterns will be regularly occurring.

2.7 Chapter Summary

During this chapter a number of video compression techniques have been discussed. These include motion compensation prediction methods, transformation, quantisation and entropy coding. All MPEG and H.26L series video compression standards are hybrid codecs that use both prediction and transformation techniques.

Motion compensated prediction algorithms are typically complex. With an increase in commercially available processing power, these methods can further be refined and improved. The evolution of both MPEG and the H.26L series of codecs is discussed from the introduction of temporal prediction through to allowing varying image block sizes for motion estimation.

Chapter 3

The H.264 Recommendation

3.1 Chapter Overview

The design overview of the H.264 Recommendation is stated in the Draft Standard (2003, p. xiv) as:

The coded representation specified in the syntax is designed to enable a high compression capability for a desired image quality. The algorithm is not lossless, as the exact source sample values are typically not preserved through the encoding and decoding processes. A number of techniques may be used to achieve highly efficient compression. Encoding algorithms may select between inter and intra coding for block-shaped regions of each picture. Inter coding uses motion vectors for block-based inter prediction to exploit temporal statistical dependencies between different pictures. Intra coding uses various spatial prediction modes to exploit spatial statistical dependencies in the source signal for a single picture. Motion vectors and intra prediction modes may be specified for a variety of block sizes in the picture. The prediction residual is then further compressed using a transform to remove spatial correlation inside the transform block before it is quantised, producing an irreversible process that typically discards less important visual information while forming a close approximation to the source samples.

Finally, the motion vectors or intra prediction modes are combined with the quantised transform coefficient information and encoded using either variable length codes or arithmetic coding.

This chapter provides information regarding the H.264 standard with respect to available profiles, specific video coding techniques adopted and includes a brief discussion on the formatting of the data prior to transmission or storage.

The H.264 standard quite simply defines an encoded video bitstream and a method of decoding this bitstream.

3.2 H.264 Profiles

The first version of the standard defined three profiles and fifteen levels. These three profiles are the Baseline, Main and Extended profiles. The Extended profile is an expansion of the Baseline profile.

Encoders are required to provide conforming bitstreams consistent with their declared profile and level. A decoder that conforms to a specific profile must be able to support all of its features. Interoperability between individual encoders or decoders is achieved through this profile and level compliance.

This research project specifically focuses on the Baseline profile which includes the following video coding tools:

1. Entropy coding - Context-adaptive variable-length coding (CAVLC),
2. I and P Slices
3. Redundant Slices
4. FMO - Flexible Macroblock Ordering
5. ASO - Arbitrary Slice Ordering

The Extended profile includes all of the techniques available within the Baseline profile as well as the following:

1. SI, SP and B Slices
2. Data Partitioning
3. Interlace frame/field coding
4. Picture adaptive frame/field
5. Macroblock adaptive frame/field

The Main profile best supports entertainment video applications. This includes digital versatile disk (DVD) or broadcast video allowing the following video compression techniques to be used:

1. Entropy coding - Context-adaptive variable-length coding (CAVLC) and
2. Entropy coding - Context-adaptive binary arithmetic coding (CABAC)
3. I, P and B Slices
4. Interlace frame/field coding
5. Picture adaptive frame/field
6. Macroblock adaptive frame/field

The fifteen levels defined in the standard vary for each profile and refer to decoder processing load, memory availability and picture size.

It is envisaged that the Baseline profile would be primarily used for video conferencing or conversational services over the internet. The Baseline profile particularly supports applications with limited data rates and low latency requirements. The Extended profile expands on the Baseline profile and supports streaming services where latency is not an issue. It seeks to provide improved fidelity.

Sullivan and Wiegand (2005) indicate that there are now four additional profiles relating to the Fidelity Range Extensions (FRExt). These profiles are High, High 10, High 4:2:2 and High 4:4:4 profiles. These new profiles address different fidelity and input image colour components for various applications. There is currently little literature available on the specifics of these new profiles and they will not be addressed in this project.

H.264 deals with input images that consist of Y, Cr and Cb components. Y is the luminance or luma component representing brightness. Cr and Cb represent the chrominance or chroma components of the image, which is the amount that red and blue deviate from grey.

Commonly the format for consumer applications such as video conferencing, digital television and DVD is 4:2:0. This means that half of the bits used to represent the image are for luminance and the other half for chrominance. The chrominance bits are further halved into their red and blue components. Less bit information is required to represent chrominance as the human eye is less sensitive to chrominance than luminance.

3.3 Bitstream Formats

There are two formats specified within the Recommendation. The Network Abstraction Layer (NAL) unit stream format and the byte stream format. The two bitstreams are essentially the same in that both formats consist of the necessary syntax structures in a pre-designated order. The byte stream format prefixes each NAL Unit (NALU) with a start code prefix and may also include a number of zero-valued bytes.

For both formats, a NALU consists of a header followed by the video coding layer (VCL) or non-VCL data. The unit stream format is a bitstream of consecutive NALUs used for packet orientated transport.

The byte stream format provides a clear and distinct demarcation method between consecutive NALUs by requiring a start code prefix. The start code is either a 3 or 4 byte code, being 0x000001. The 4 byte code, or long start code is required for NALUs containing sequence parameter sets, picture parameter sets and the first VCL of a

picture. All other NALUs may use at least a 3 byte start code. Any number of zero bytes may be used prior to a NALU and once the start code is detected by the decoder it is discarded.

The encoder bitstream used as an example in this paper uses the byte stream format as shown in Appendices C, D and E.

An access unit is the set of VCL and non-VCL NAL units that are associated with a single decoded picture. An access unit may also contain an end of stream or end of sequence NAL unit which indicates the completion of the stream or video sequence.

3.4 Network Abstraction Layer

The NAL formats the encoded video and provides header information to package data for transport. The packaged data may then be stored or passed to a transport protocol for encapsulation and transmission. The NAL provides the interface between the video codec itself and the outside world. A NALU is the individual unit used to provide the generic formatting for the data.

NALUs can primarily be divided into two types, VCL and non-VCL. Table B.2 displays the different kinds of data that make up each of these types.

The VCL deals directly with the actual video content and divides each picture into slices and macroblocks to allow video compression to occur.

The non-VCL NALUs provide extra information such as the number of frames, frames skipped and specific video compression techniques required for decoding. Non-VCL NALUs may provide additional information relating to the entire video sequence or a single frame.

The first byte of all NAL packets is a header byte and provides specific header information including an indication of the type of data contained in the rest of the packet. This allows the decoder to determine the requirements of the incoming bitstream.

A finite state machine also referred to as an entropy encoder ensures that start code prefixes used prior to each NALU in the byte stream format are not accidentally emulated within the completed NALU bitstream. This process is further investigated in Chapter 4.

3.4.1 Data Partitioning

Data partitioning is allowed only in the Extended profile. This method is used to increase the robustness of the video data. Data partitioning allows for intra, inter and header information to be separated into their own partitions. Each data partition may be placed into separate NALUs for transmission or storage.

3.5 Slices

There are five slice types supported by H.264, being:

I slices - All the macroblocks of the I or Intra slice are coded using intra prediction. An instantaneous decoding refresh (IDR) picture contains only I slices.

P slices - Macroblocks of a P slice can be coded using intra or inter prediction. P or Predictive slices allow only one MCP signal per macroblock to be used

B slices - Like P slices, the macroblocks of a B or Bipedictive slice can be coded using intra or inter prediction, however two MCP signals may be used per prediction and if so are combined using a weighted average.

SP slice - This allows P slices to be switched between different video streams efficiently. An advantage of SP or Switching P and SI or Switching I slices is the ability to use trick modes, such as fast-forward or fast reverse.

SI slice - The SI slice is an exact match for an SP slice for random access or error recovery, while using only Intra Prediction.

Redundant Slices are also used to duplicate coded representations of all or some parts of an image in order to increase the robustness of the code.

3.6 Motion Compensated Prediction

Intra Prediction

There are 3 types of intra coding supported by the H.264 Recommendation, Intra_4×4 prediction, Intra_16×16 prediction and I_PCM.

Intra_4×4 prediction is commonly used when there is significant detail in the picture. Intra_4×4 predicts the sixteen 4×4 luma blocks within one macroblock individually, whereas 16×16 mode predicts the entire 16×16 luma block. A separate chroma block prediction is also conducted for each type, however chroma prediction only occurs with the entire 16×16 macroblock. Intra_4×4 prediction supports nine spatial prediction directions whilst Intra_16×16 supports four.

The third type of intra coding is I_PCM where raw values of the macroblock are sent without prediction or transformation. This ensures that the number of bits needed for an encoded macroblock will not be greater than the number required of an uncompressed macroblock. This value is determined irrespective of the quantization step size.

Inter Prediction

Inter prediction uses motion compensation in order to predict the macroblocks. Inter prediction supports 16×16, 16×8, 8×16 and 8×8 block sizes.

H.264 allows smaller block sizes by using a tree structured form of partitioning the macroblocks. Should the 8×8 block size be chosen, the block may be further divided into 8×4, 4×8 or 4×4. The use of a smaller block size will produce a smaller encoded frame and require additional motion vectors which may negate the positive effects

gained from using a smaller block size.

The standard uses a sum of absolute differences (SAD) approach for computing an appropriate block size. Each 4×4 block is predicted for the best match and the SAD values for adjacent blocks are added to produce bigger blocks. This method reduces the requirement to test all of the possible H.264 block size combinations.

H.264 allows quarter-pel interpolation for motion estimation. This motion compensation method finds the best match complete block and seeks to refine the estimation by moving the block by half or a quarter of the block size around the best match block.

The Main profile also allows for weighted prediction, bipredictive and direct prediction motion compensation methods. Bipredictive motion compensation seeks to use the average of two reference pictures for the prediction values of the current block. Weighted prediction scales the reference pictures. This method will also allocate each picture a value and when combined will equal one. This value therefore determines how much influence each reference picture has on the current prediction value. Direct prediction does not pass any prediction information, residual frame or motion vector to the decoder. The decoder will perform the prediction by predicting the motion vector information based on previous encoded pictures.

3.7 Flexible Macroblock Ordering

Flexible macroblock ordering (FMO) may be used to change the way macroblocks are associated with slices. Macroblocks may be sent in a more flexible and efficient manner using FMO as they are mapped to specific slice groups as opposed to being mapped as consecutive macroblocks using raster scan.

H.264 allows macroblocks within pictures to use a variety of slice mapping patterns. These include interleaving, dispersed, foreground groups, box-out, wipe, explicit and raster scan. Slice groups may also be mapped in inverse raster scan, wipe left and counter clockwise box out directions. Figure 3.1 diagrammatically represents the available slice group maps.

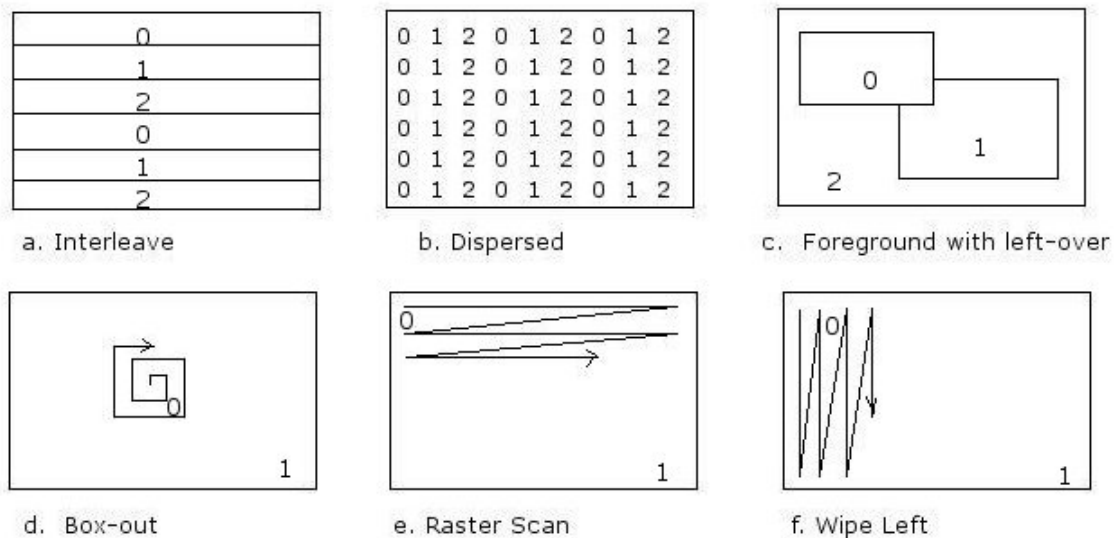


Figure 3.1: Slice group maps available for use with Flexible Macroblock Ordering.

3.8 Arbitrary Slice Ordering

Arbitrary slice ordering (ASO) allows slices to be decoded in a different order than their designated display order. ASO improves upon loss robustness and delay reduction. ASO is particularly important for real-time applications or for networks which may commonly deliver data out of order.

Without ASO slices must allow the first macroblock of each slice to be in the correct order for the raster scan of the picture. The raster scan of a picture requires that the macroblocks begin at the top left proceeding left to right, and ending at the bottom right macroblock of that picture.

3.9 Integer Transform

H.264 uses an integer transform that is less complex than previous codecs and may be implemented by 16-bit processors. The integer transform is applied to all 4×4 blocks and as defined will produce identical decoded video as originally transformed.

There are three transforms that may be applied in H.264. The first as previously

described is the integer transform that is applied to all 4×4 blocks. If the macroblock is predicted using Intra_16 \times 16, a second 4×4 transform is applied. This Hadamard transform is applied for the processing of the luma components and 2×2 Hadamard transform is applied to the chrominance components.

3.10 Logarithmic Quantisation

A quantisation parameter (QP) determines the quantisation or scaling of the transform coefficients in H.264. The QP can take on 52 values. As the QP is determined logarithmically the step size will double with each increment of six of the QP. The QP is controlled by the encoder's rate control algorithm.

3.11 Entropy Encoding

There are two types of entropy encoding supported by H.264. These being context-adaptive variable-length coding (CAVLC) and context-adaptive binary arithmetic coding (CABAC). CABAC coding is only supported in the Main profile of H.264.

Both entropy coding types use Exp-Golomb code. Exp-Golomb code is defined in the Draft Standard (2003, p149).

CAVLC entropy coding uses a total of 32 different VLC values which are mapped from a 4×4 block of image coefficients following quantisation, transformation and prediction. A total of 32 different VLCs are used in CAVLC entropy coding mode. There are four VLC tables and the table chosen is dependant upon the coefficients' values. Three of the VLC tables are context conditional and therefore the coding efficiency is better than for schemes using a single VLC table.

CABAC uses binary arithmetic coding allowing a noninteger number of bits to be assigned to each symbol of the alphabet. CABAC uses rules based tables that are not required to be stored. CABAC entropy coding uses probability models for symbol occurrence and may use a single codeword for a string of symbols.

The major difference between these two methods lies in the mapping of the Exp-Golomb codewords. CABAC coding is more complex than CAVLC, however bit rates are reduced by 10%- 15% when compared with the same quality video coded with CAVLC (Sullivan and Wiegand, 2005).

3.12 Adaptive Deblocking Filter

H.264 defines an adaptive deblocking filter for the decoder to reduce the blockiness of the image that results from block based encoding. The filtering process is conducted in a loop which improves the visual quality as required by the Recommendation. The deblocking filter is defined for use with raster scan ordering of macroblocks. Frames may need to be reordered prior to display depending on the FMO in use.

The deblocking filter may be adjusted with respect to the filtering strength. This ranges from zero to four, where zero equates to no filtering and four infers strong filtering.

3.13 Video Formats

A video frame contains two fields being the top field which has all of the even numbered rows of an image and the bottom field which contains the odd numbered rows of the image. This type of video frame is called interlaced video and the use of two fields aids in the error resilience of the video sequence.

Interlaced video uses two sampling periods to capture one full frame of video, one sampling period captures the top field and the next sampling period captures the bottom field of the subsequent image. For the coding of interlaced video, H.264 supports two coding modes, being frame and field mode.

Frame Mode

Progressive video format requires that both fields are to be captured during each sampling period. Progressive video format is also referred to as frame mode and is the only video formatting mode allowed by the Baseline Profile. The two fields of one frame are coded together in frame mode.

Field Mode

The two fields of a frame are encoded separately for field mode and is a technique that may be employed by both the Main and Extended Profiles.

These two different coding modes may be selected for each image or for each macroblock. If they are selected for each image, it is known as Picture-adaptive frame/field (PAFF), whereas if selected at macroblock level, it is referred to as Macroblock-adaptive frame/field mode (MBAFF). Neither PAFF and MBAFF may be used for the Baseline Profile.

Frame mode is good for regions that are not moving. If there are moving regions it is more efficient to use field or PAFF mode and code the fields separately.

3.14 Hypothetical Reference Decoder

The Hypothetical Reference Decoder (HRD) is employed as part of the encoder to ensure a conforming H.264 bitstream is generated. Capacity constraints of the coded picture buffer (CPB) and decoded picture buffer (DPB) are specified by the decoder. The CPB is used to assess the timing of the coded bits and the DPB is used for the storage of decoded pictures. The HRD allows for transmission of the video information at a variety of bit rates and with regard to latency issues. The DPB buffer should also be large enough to allow for multipicture buffering.

3.15 Chapter Summary

The H.264 Recommendation defines three profiles being Baseline, Main and Extended. Video compression techniques specifically allowed by the Baseline profile include CAVLC entropy coding, I and P Slices, redundant slices, FMO and ASO.

H.264 allows 4×4 blocks for motion compensated prediction and quarter pel interpolation. An efficient integer transform replaces a discrete cosine transform that was commonly used by previous video compression standards. Logarithmic quantisation is defined by the standard and an adaptive deblocking filter is introduced to improve the visual integrity of the video.

Chapter 4

H.264 Encoder Principles

4.1 Chapter Overview

The H.264 encoder principles are discussed in length within this chapter. The first block of a raw video sequence is followed through the encoding process in detail, specifically to highlight various compression techniques. The H.264 standard does not define an encoder, however, the provision of a conforming bitstream and methods to decode this bitstream are defined.

This chapter highlights the necessity of employing sequence and picture parameter sets in order to achieve a successfully encoded bitstream.

Reference to particular sections of the encoder source code have been made to show practical algorithms of H.264 video compression techniques. The encoder source code has been derived from the public domain reference software that has been made available by Dr Karsten Suehring and the Joint Video Team (2004) with significant changes made in order to optimise the code for the Baseline Profile.

The chapter follows the encoding process of a source frame from the container_qcif.yuv video sequence resourced from the Kansas State University website. The first block of pixel values from this frame is following through the encoding process.

4.2 The Encoder

A typical H264 encoder diagram is shown in Figure 4.1.

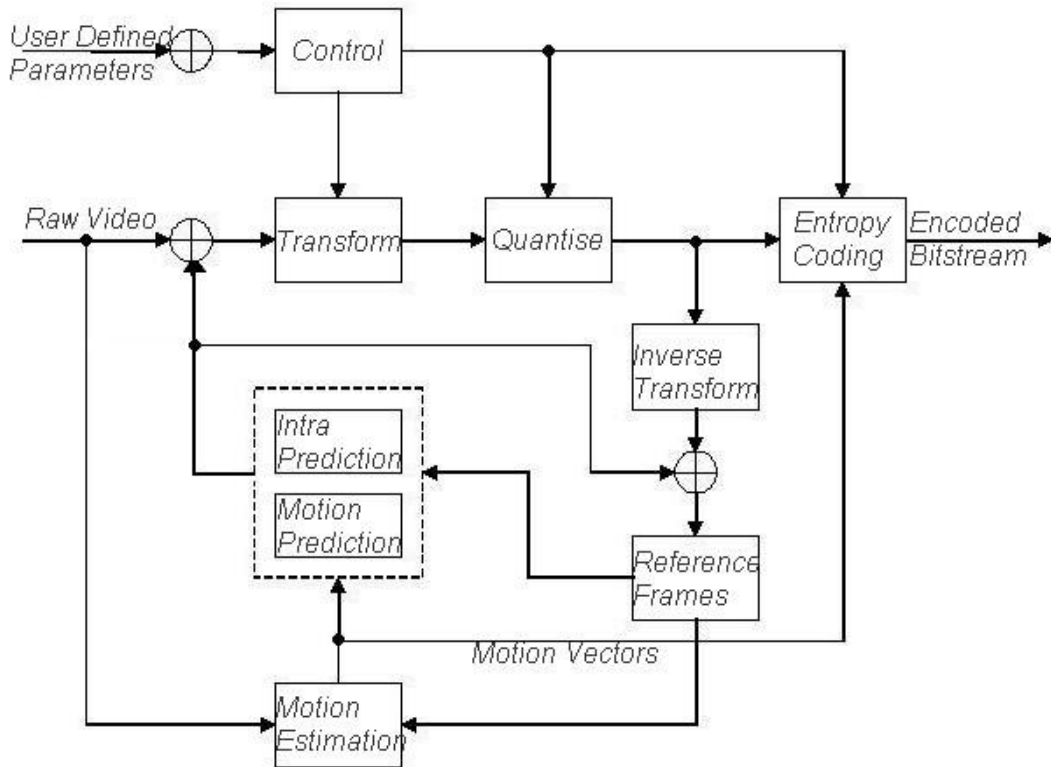


Figure 4.1: H.264 Encoder Block Diagram

User defined parameters required for encoding a raw video sequence are used as control parameters and directly determine the values employed by the various video compression techniques.

The raw video sequence is used for motion estimation comparisons and the resulting motion compensation predictions are integer transformed. Depending upon the user defined parameters, may also be Hadamard transformed.

The transformed coefficients are quantised and entropy encoded prior to transmission or storage. The hypothetical reference decoder will inverse transform the frames at the encoder and store the decoded frames to memory for future motion estimation requirements.

4.3 The Encoded Bitstream

The exact bitstream output from a H.264 compliant encoder is determined by not only the incoming raw video, but the controlling instructions on how this raw video is to be treated. The semantics that need to be provided to the encoder include the output bitstream format, number of slices to use, macroblock mapping and quantisation parameter.

The method by which the image data will be compressed and the quality of the resulting decoded image is a result of not only the video compression techniques but the level that they will be employed. This information is required to be passed to the decoder as part of the encoded bitstream, to ensure that correct decoding techniques are employed.

The user of the research project software provides the necessary information to the encoder software through the Graphical User Interface (GUI). Specific encoder parameters are made available to the user for changing. These parameters are written to a text file, called `encoder.cfg`. The encoder executable will read this file to allow the parameter values to be employed in the encoding process. The `encoder.cfg` file is available to be read by the user to revisit the last employed parameter values.

The encoder software stores all of the `encoder.cfg` parameters into the Input Parameters structure called `input`. The input structure provides a storage and recall mechanism for the user defined values. Image parameters are created directly from the input parameters and the raw video bitstream.

The image parameters structure is used by the encoder for the values to be applied for all of the necessary encoding algorithms, as well as detailing storage for coefficients during various stages of compression. A portion of the `ImageParameters` structure is listed below and has been commented in detail to highlight it's importance to the software's encoding process.

Listing 4.1: The ImageParameters structure listing may be found from Line 651 to Line 845 of Encoder header file global.h

```

typedef struct
{
    int number;                //!< current image number to be encoded
    int pn;                    //!< picture number
    int current_mb_nr;         //!< current macroblock number
    int total_number_mb;      //!< total number of macroblocks of image
    int current_slice_nr ;    //!< current slice number within image
    int type;                  //!< I frame (1) or P frame (0)
    int structure;            //!< picture structure (Frame / Field)
    int num_ref_frames;        //!< number of reference frames
    int max_num_references;    //!< max number of ref pictures – encoder.cfg
    int qp;                    //!< quant for the current frame – encoder.cfg
    float framerate;          //!< Frames per second – encoder.cfg
    int width;                 //!< Number of pels – encoder.cfg
    int height;                //!< Number of lines – encoder.cfg
    int subblock_x;           //!< current subblock horizontal
    int subblock_y;           //!< current subblock vertical
    int mb_y_upd;              //!< number of intra macroblocks per frame

    int block_c_x;            //!< current block chroma vertical
    int **ipredmode;          //!< intra prediction mode

    int cod_counter;          //!< Current count of skipped macroblocks in a row
    int ***nz_coeff;          //!< number of coefficients per block (CAVLC)

    int mb_x;                  //!< current MB horizontal
    int mb_y;                  //!< current MB vertical
    int block_x;               //!< current block horizontal
    int block_y;               //!< current block vertical
    int pix_x;                 //!< current pixel horizontal
    int pix_y;                 //!< current pixel vertical
    int pix_c_x;               //!< current pixel chroma horizontal
    int pix_c_y;               //!< current pixel chroma vertical

    int opix_x;                //!< current original picture pixel horizontal
    int opix_y;                //!< current original picture pixel vertical
    int opix_c_x;              //!< current original picture pixel chroma horizontal
    int opix_c_y;              //!< current original picture pixel chroma vertical

    imgpel mpr [9][16][16];    //!< all 9 prediction modes
    imgpel mpr_2 [5][16][16];  //!< all 4 new intra prediction modes
    imgpel mpr_c [2][4][16][16]; //!< chroma intra prediction modes
    imgpel mpr [16][16];       //!< current best prediction mode
    int m7 [16][16];           //!< diff pixel values btn orginal image and prediction

    int ****cofAC;             //!< AC coeffs [8x8block][4x4block][level/run][scan_pos]
    int ****cofDC;             //!< DC coeffs [yuv][level/run][scan_pos]

    Picture *currentPicture;   //!< The current coded picture
    Slice *currentSlice;       //!< pointer to current Slice data struct
    Macroblock *mb_data;       //!< array containing all MBs of a whole frame
    SyntaxElement MB_SyntaxElements[MAX_SYMBOLS_PER_MB]; //!< temp MB se

    int *quad;                 //!< Array of square values,used for snr computation
    int***** pred_mv;         //!< motion vector predictors for blocks and ref frames

```



```

int LFDisableIdc;           ///< Disable deblocking loop filter
int LFAlphaCOffset;        ///< Loop Filter Alpha Offset
int LFBetaOffset;          ///< Loop Filter Beta Offset

int num_ref_idx_l0_active;  ///< reference frame list
int nal_reference_idc;      ///< sent in slice header – type of NALU
int DeblockCall;           ///< Used to indicate if Deblocking was performed
int pre_frame_num;         ///< previous frame number
int slice_group_change_cycle; ///< Slice Group Change Rate – encoder.cfg

int pic_unit_size_on_disk;  ///< used with bitdepth_luma and chroma
int bitdepth_luma;          ///< Brightness – encoder.cfg
int bitdepth_chroma;        ///< Colour – encoder.cfg
unsigned int dc_pred_value; ///< value for DC prediction
int max_imgpel_value;       ///< max value that one picture element (pixel)
int max_imgpel_value_uv;    ///< max value for one chroma picture element

int yuv_format;            ///< Raw YUV format – encoder.cfg
int mb_cr_size_x;           ///< mb chroma size – x direction
int mb_cr_size_y;           ///< mb chroma size – y direction
} ImageParameters;

```

4.3.1 Parameter Sets

There are two parameter set types employed by H.264, being the Sequence Parameter Set (SPS) and the Picture Parameter Set (PPS). A parameter set contains header information that can apply to a large number of video coding layer (VCL) network abstraction layer units and rarely changes.

The encoding of a picture requires both parameter sets. A SPS applies to an entire video sequence, whereas a PPS applies to one or more pictures within that sequence. Each picture within the sequence will specify which PPS it belongs to and each PPS will specify an SPS.

Both parameter sets have their own NAL units (NALUs) which may be transmitted separately to the VCL NALUs of the video sequence for error resilience purposes. If the parameter sets are to be transmitted separately, a new data partition will need to be created, however, only one partition is allowed in the Baseline Profile. The parameter sets will be incorporated into this partition and will not be segregated for transmission

or storage.

Appendices C and D explain the syntax elements of the SPS and PPS respectively. These Appendices highlight the interaction of these syntax elements with the encoding process and their inclusion into the bitstream. The specific formatting of the parameter sets into NALUs is provided in Appendix B and the byte stream format for both parameter sets is shown in Appendices C and D.

Figure 4.1, the encoder block diagram, places the parameter sets into the control block. Although not depicted in Figure 4.1 the control block provides the necessary variables to all of the other blocks on the diagram.

For the baseline profile the sequence parameter set, followed by the picture parameter set are the first the first two NALUs to be placed onto the bitstream. The third NALU to be placed onto the bitstream is the encoded intra picture data.

4.3.2 Intra Video Coding Layer Network Access Layer Unit

There are two types of encoding image data in the H.264 Recommendation, being intra and inter prediction. An intra predicted frame is a frame that is predicted without reference to any other frame. An inter predicted frame references other frames for motion prediction, and although this increases the complexity of the encoding process, the required file size is dramatically reduced.

An instantaneous decoding refresh (IDR) access unit is an intra picture that signals that no future frames will reference any frames prior to the IDR picture. An IDR picture may be used to ensure that errors are not propagated throughout the entire video sequence, provide suitable entry points into the video stream and is therefore the first encoded frame of a video sequence.

Source Video Frame

The first frame is read in from the `container_qcif.yuv` source file. The raw video uses 25344 bytes for defining the luma component and a further 6336 bytes for each of the u and v chroma components of the frame. The raw video is therefore using YUV 4:2:0 format, as 6336 is a quarter of 25344, and there are two groups of 6336 bytes, one for each of the chroma components. The original source frame is shown in Figure 4.2, sourced from Kansas State University(n.d).



Figure 4.2: The original first frame from the `container_qcif.yuv` video sequence

The length of the luma component, 25344 bytes, for a raw video frame is determined by the horizontal pixel size of the source frame multiplied by the vertical pixel size and the symbol size in bytes. The source frame dimensions are derived from the user input provided in the encoder graphical user interface, GUI. Default values of 176 picture elements for image width and 144 picture elements for the image height, as per the Quarter Common Intermediate Format, QCIF, have been used for the encoder software.

The length of the chroma component may be determined by multiplying half of the horizontal size of the source frame by half of the vertical size, and multiplying this by the symbol size in bytes, equating to a quarter of the number of bytes required for the luma component.

The source frame structure used in the encoder software may be found at L462 of

global.h. The program listing is shown below.

Listing 4.2: The source frame structure listing

```
typedef struct
{
    // Size information
    int x_size, y_framesize, y_fieldsize ;
    int x_size_cr, y_framesize_cr, y_fieldsize_cr ;
    imgpel *yf, *uf, *vf;           // frame representation
    imgpel *yt, *ut, *vt;         // top field
    imgpel *yb, *ub, *vb;         // bottom field
} Sourceframe; // Used for the original yuv picture information.
```

The luma component is stored in the source frame structure's yf array. The u and v chroma components are stored in the uf and vf arrays respectively. The luma and chroma components are then split into top and bottom fields of the frame.

The other parameters of this structure are determined by the frame format and the YUV formatting used.

This part of the process is simply represented by the raw video input arrow of Figure 4.1, the encoder block diagram.

Preparing to Code a Picture

In preparation for encoding the video sequence, temporary memory is allocated to a bitstream buffer to store raw video and for motion prediction.

The flexible macroblock ordering, (FMO), map is organised based on the user selection, depicted in Figure 6.6. The map used for encoding of the container_qcif frame uses no FMO, whereby a map consisting entirely of zeros is used and macroblocks are allocated to one slice group in raster scan order, as shown in Figure ???. Macroblock mapping occurs prior to the start of encoding of all frames.

A memory allocation of 64000 bits is prepared for the outgoing NALU. The Slice header is first to be placed into the bitstream buffer. The Slice header syntax elements include:

current_mb_nr equal to 0, due to this being the first macroblock of the slice, with counting starting at 0;

slice_type indicates that the slice is an I slice;

pic_parameter_set_id designates which picture parameter set was used to encode the slice;

frame_num frame slice belongs to;

idr_pic_id indicates that this is an IDR picture;

pic_order_cnt_lsb derived by the number of frames to be encoded multiplied by the number of frames to skip times the number of fields to be encoded;

no_output_of_prior_pics_flag used for the decoded picture buffer (DPB);

long_term_reference_flag used to ensure current IDR picture is used for long term reference;

slice_qp_delta used with the user defined quantisation parameter;

The slice parameter information is described in Appendix B. Inclusion of the slice header into the bitstream is shown in Appendix E for the example container.qcif frame.

Selecting the most efficient prediction method

The H.264 Recommendation details the requirement to select the most efficient prediction method. All of the possible prediction methods need to be tested on the original image in order to determine the method with the least cost. When encoding the first macroblock of the first slice and frame the only available prediction method is DC prediction, referred to in the software as DC_PRED_8, or mode 0. Each of the predicted values are equal to 128 and the midpoint value of the available range.

The total cost of the original image data is derived using the LPCM method. This method is treated as a prediction method in order to confirm that the total cost of the predicted macroblock is less than that required for the original data.

Figure 4.3 is an expanded view of the top left corner of the first container ship frame containing 16 macroblocks.

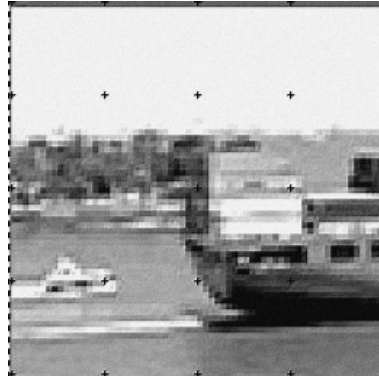


Figure 4.3: A magnified 4×4 block of macroblocks of the top left corner of the first frame of the original YUV video.

DC_PRED_8 divides the macroblocks into blocks of 8×8 , and each 8×8 block has one 4×4 chroma u block and one 4×4 chroma v block. This 8×8 block will also have four 4×4 block luma blocks.

The intra prediction of the chroma values of the macroblock is conducted first. The individual differences between the u chroma values of the source image and the chroma prediction values, determined to be 128, for each pixel of the first 4×4 block is determined. The first four each have a value of 135, therefore a difference of 7 will result. The differences for this u chroma block are shown in Table 4.1.

	uc0	uc1	uc2	uc3
ur0	7	7	7	7
ur1	9	9	10	10
ur2	9	9	9	9
ur3	9	9	9	9

Table 4.1: Differences between original u chroma values and best mode predicted values.

These differences in conjunction with whether a Hadamard transform is used determine the cost associated with this first 4×4 chroma block. If Hadamard transform is not used the absolute value of each of the differences is simply added, to derive a cost of 138.

The next three 4×4 u chroma blocks are also costed in this way, as well as the first four 4×4 pixel blocks of v chroma values. The subsequent addition of all of these eight costs

is used to determine the best method for chroma prediction including a comparison with original data costs.

Following u and v chroma prediction, allows motion estimation to be conducted on the luma component. The original luma values for the first 4×4 block is shown below in Table 4.2.

	C0	C1	C2	C3
R0	106	106	106	106
R1	255	255	255	255
R2	231	231	230	231
R3	230	231	231	232

Table 4.2: The first 4×4 block of luma values of the raw container_qcif.yuv video.

Comparing Table 4.2 with Figure 4.3 shows the luma difference between the first and second row of the image and the large numerical difference between their respective values.

The difference between the original Y or luma component of the source image and the predicted value of 128 is evaluated for this 4×4 block. Table 4.3 shows these difference values. These values are stored in the m7 array, see Listing 4.3, ready for transformation.

	i0	i1	i2	i3
j0	-22	-22	-22	-22
j1	127	127	127	127
j2	103	103	102	103
j3	102	103	103	104

Table 4.3: Calculated luma differences of the first 4×4 block.

The total cost of this frame is calculated to be 1419 which is less than the cost of the original 4×4 block being 2491.

Transforming the efficiently predicted block

The differences between each of these luma pixels and the prediction value is integer transformed in the encoder function `dct_luma`. This function provides several of the compression techniques employed in this recommendation including quantisation of the transformed coefficients prior to entropy coding and reestablishing a decoded frame for referencing. This function is shown in Figure 4.1 as the area encompassing transform, quantise, inverse transform, adding the prediction and storing the reference frame.

Earlier video compression standards employed a lossy discrete cosine transform (DCT). H.264 introduces a lossless integer transform that uses different values. However the transform is conducted using the same method. The matrix used for the integer transform is detailed in the Draft Standard (2003, p. xiv) and is shown below in Table 4.4.

	i0	i1	i2	i3
j0	1	1	1	1
j1	2	1	-1	-2
j2	1	-1	-1	1
j3	1	-2	2	-1

Table 4.4: The Integer Transform matrix used for 4×4 blocks.

Figure 4.4, is sourced from Malvar et al (2003) and is a graphical representation of the transformation process. This diagram displays the interaction between the difference values to produce the transform coefficients.

The integer transform is conducted in two parts. The horizontal transform is conducted followed by the vertical transform. Listing 4.3 shows the transformation section of the `dct_luma` function.

Listing 4.3: Transformation section of `dct_luma` function.

```
// Horizontal transform
for (j=0; j < BLOCK_SIZE && !lossless_qpprime; j++)
{
  for (i=0; i < 2; i++)
  {
    i1=3-i;
    m5[i]=img->m7[i][j]+img->m7[i1][j];
```

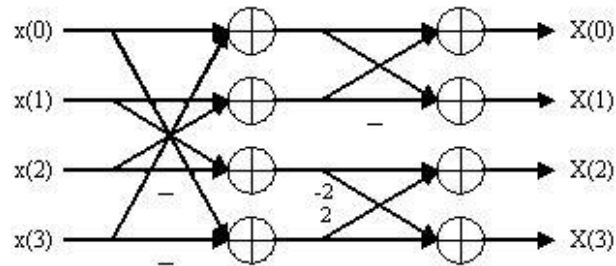



Figure 4.4: Integer Transformation Method.

```

    m5[i1]=img->m7[i][j]-img->m7[i1][j];
}
img->m7[0][j]=(m5[0]+m5[1]);
img->m7[2][j]=(m5[0]-m5[1]);
img->m7[1][j]=m5[3]*2+m5[2];
img->m7[3][j]=m5[3]-m5[2]*2;
}
// Vertical transform
for (i=0; i < BLOCK_SIZE && !lossless_qprime; i++)
{
    for (j=0; j < 2; j++)
    {
        j1=3-j;
        m5[j]=img->m7[i][j]+img->m7[i][j1];
        m5[j1]=img->m7[i][j]-img->m7[i][j1];
    }
    img->m7[i][0]=(m5[0]+m5[1]);
    img->m7[i][2]=(m5[0]-m5[1]);
    img->m7[i][1]=m5[3]*2+m5[2];
    img->m7[i][3]=m5[3]-m5[2]*2;
}

```

The result of the horizontal transform is displayed in Table 4.5. These values become the input `img->m7` array to the vertical transform.

	i0	i1	i2	i3
j0	-88	0	0	0
j1	508	0	0	0
j2	411	1	1	-2
j3	412	-4	0	-2

Table 4.5: The 4×4 block following horizontal transform.

The vertical transform uses a transposed version of the integer transform matrix, see Table 4.4, to calculate the vertical transform coefficients. Table 4.6 below shows the results of the vertical transform.

	i0	i1	i2	i3
j0	1243	-3	1	-4
j1	-903	7	-1	6
j2	-595	-5	-1	0
j3	-694	6	2	-2

Table 4.6: The 4×4 block of transformed coefficients.

The transformed coefficients are now be quantised.

Quantisation

Quantisation now occurs on the transformed coefficients. Listing 4.3.2 provides the relevant source code reproduced from the `dct_luma` function from the encoder software.

Listing 4.4: Quantisation code listing of `dct_luma` function.

```

qp_per = (currMB->qp + img->bitdepth_luma_qp_scale - MIN_QP)/6; // 6
qp_rem = (currMB->qp + img->bitdepth_luma_qp_scale - MIN_QP)%6; //Remainder of 6
q_bits = Q_BITS+qp_per; //increase binary representation from 15

if (img->type == I_SLICE)
    qp_const=(1<<q_bits)/3; // intra (2^q_bits) / 3
else
    qp_const=(1<<q_bits)/6; // inter

// Quant
nonzero=FALSE;
run=-1;
scan_pos=0;

for (coeff_ctr =0;coeff_ctr < 16;coeff_ctr++)
{
    i=SNGL_SCAN[coeff_ctr][0];
    j=SNGL_SCAN[coeff_ctr][1];

    run++;
    ilev=0;

    if(intra == 1)
        level=(abs(img->m7[i][j])*LevelScale4x4Luma_Intra[qp_rem][i][j]+qp_const)>>q_bits;
    else
        level=(abs(img->m7[i][j])*LevelScale4x4Luma_Inter[qp_rem][i][j]+qp_const)>>q_bits;

```

```

if (level != 0)
{
  nonzero=TRUE;
  if (level > 1 || lossless_qpprime)
    *coeff_cost += MAX_VALUE;           // set high cost
  else
    *coeff_cost += COEFF_COST[0][run];
  ACLevel[scan_pos] = sign(level, img->m7[i][j]);
  ACRun [scan_pos] = run;
  ++scan_pos;
  run=-1;                               // reset zero level counter

  level=sign(level, img->m7[i][j]);
  if (qp_per<4)
  {
    if (intra == 1)
      ilev=(level*InvLevelScale4x4Luma_Intra[qp_rem][i][j]+
(1<<(3-qp_per)))>>(4-qp_per);
    else
      ilev=(level*InvLevelScale4x4Luma_Inter[qp_rem][i][j]+
(1<<(3-qp_per)))>>(4-qp_per);
  }
  else
  {
    if (intra == 1)
      ilev=(level*InvLevelScale4x4Luma_Intra[qp_rem][i][j])<<(qp_per-4);
    else
      ilev=(level*InvLevelScale4x4Luma_Inter[qp_rem][i][j])<<(qp_per-4);
  }
  }
  img->m7[i][j]=ilev;
}

```

The number of quantisation bits used for the luma component is defined by the `q_bits` variable in Listing 4.4. Variables of this listing are defined below:

currMB->qp user defined Quant. for First Frame or Quant. for Remaining Frames as described in Section 6.3.

img->bitdepth_luma_qp_scale user defined Brightness variable defined later in Section 6.3.

MIN_QP The minimum quantisation parameter is defined to be 8 bits.

qp_per Values used in this example become $(16 + 8 - 8)/6 = 2$ with a remainder of 4.

qp_rem This variable uses the remainder value from the evaluation of `qp_per`.

q_bits `Q_BITS` is defined to be 15.

qp_const The intra qp_const employed during this chapter is determined to be $(2^{q_bits})/3$ which equals 43690.

The quantisation process may be followed for the first value of Table 4.6, 1243. The transformed coefficient is multiplied by a quantisation value determined by pixel position and qp_rem. The quantised coefficient is $((1243*8192)+43690)/2^{17}$ equalling 78.

The logarithmic quantisation employed by H.264 is depicted in the calculation of the qp_per variable where each increase in the user defined quantisation parameter of 6, results in a doubling of the step size.

The SNGL_SCAN array instructs the accessing of coefficients within the 4×4 block to be conducted in the correct sequence. LevelScale4x4Luma_Intra contains a predetermined array of quantised coefficients.

Table 4.7 shows the transformed coefficients following quantisation.

	i0	i1	i2	i3
j0	78	0	0	0
j1	-36	0	0	0
j2	-37	0	0	0
j3	-28	0	0	0

Table 4.7: The 4×4 block of quantised values.

Replicating the Decoder

The quantised values are rescaled, inverse transformed and stored in an enc_picture array. This array allows the encoder to exactly replicate the decoder and utilise differences imposed on the data as a result of the compression techniques.

The quantised value is scaled using a dequantisation coefficient determined using the same positional details as previously used. Using a dequantisation coefficient of 256, the first dequantised value is calculated by $(256*78+2)/4$, equalling 4992. The resulting 4×4 block of dequantised values is shown in Table 4.3.2.

	i0	i1	i2	i3
j0	4992	-2880	-2368	-2240
j1	0	0	0	0
j2	0	0	0	0
j3	0	0	0	0

Table 4.8: 4×4 block of dequantised values.

These values further undergo an inverse horizontal integer transform, and the 4×4 block appears as below.

	i0	i1	i2	i3
j0	4992	4992	4992	4992
j1	-2880	-2880	-2880	-2880
j2	-2368	-2368	-2368	-2368
j3	-2240	-2240	-2240	-2240

An inverse vertical integer transform is also performed to complete the transformation process. Table 4.8 shows the resulting 4×4 block and are the values anticipated at the encoder.

	I0	I1	I2	I3
J0	107	107	107	107
J1	255	255	255	255
J2	231	231	231	231
J3	232	232	232	232

Table 4.9: 4×4 block of inverse transformed values.

These values may be checked against the original values as shown in Table 4.7 to show that minimum losses have occurred.

If another frame had been referenced the difference between the prediction and the determined value would also be added. The complete transformation, quantisation and decoder representation is continued until an intra 4×4 Macroblock consisting of 16×16 pixels has been coded.

Intra Prediction 16×16 is now conducted to determine whether this prediction method is of less cost than Intra 4×4 prediction.

Prediction, horizontal and vertical integer transforms are now performed on both the chroma u and v components. They are then quantised and inverse transformed. The 4×4 table below shows the reproduced values for the u chroma component.

	I0	I1	I2	I3
J0	137	137	137	137
J1	137	137	137	137
J2	137	137	137	137
J3	137	137	137	137

The encoded v chroma component is shown below.

	I0	I1	I2	I3
J0	126	126	126	126
J1	126	126	126	126
J2	126	126	126	126
J3	126	126	126	126

The quantised value shown in Table 4.7 will now have the motion vectors calculated.

Calculation of Motion Vectors

Due to this being the first macroblock of the first frame there are no motion vectors to predict and the motion vector arrays are coded to 0.

The quantised values then undergo entropy encoding. The only entropy encoding method available in the baseline profile is context adaptive variable length coding, CAVLC.

CAVLC Entropy Encoding

Entropy encoding is utilised to perform statistical redundancy checks and to ensure start codes are not emulated within the encoded image bitstream.

The macroblock header is placed onto the Bitstream immediately following the slice header. Macroblock header parameters are provided in Appendix B. Specific header data and the encoded header bitstream are shown in E.

The entropy encoding procedure is discussed below.

The Number of Nonzero Coefficients (nnz) for Luma Blocks is sought, with the values for the first macroblocks luma and chroma components set to 0.

The VLC code table used is dependant upon the nnz value. If the nnz value is less than 2 table 0 is used. Listing 4.5 shows a portion of the length and code tables used to determine the vlc values to be employed for NNZ coefficients.

Listing 4.5: Portion of VLC tables for length and code for NNZ use.

```

int writeSyntaxElement_NumCoeffTrailingOnes(SyntaxElement *se, DataPartition *this_dataPart)
{
    static const int lentab [3][4][17] =
    {
        { // i.e. lentab [0][0][4] = 10
          { 1, 6, 8, 9,10,11,13,13,13,14,14,15,15,16,16,16,16},
          { 0, 2, 6, 8, 9,10,11,13,13,14,14,15,15,15,16,16,16},
          { 0, 0, 3, 7, 8, 9,10,11,13,13,14,14,15,15,16,16,16},
          { 0, 0, 0, 5, 6, 7, 8, 9,10,11,13,14,14,15,15,16,16},
        },
    };

    static const int codtab [3][4][17] =
    {
        {
          { 1, 5, 7, 7, 7, 7,15,11, 8,15,11,15,11,15,11, 7,4},
          { 0, 1, 4, 6, 6, 6, 6,14,10,14,10,14,10, 1,14,10,6},
          { 0, 0, 1, 5, 5, 5, 5, 5,13, 9,13, 9,13, 9,13, 9,5},
          { 0, 0, 0, 3, 3, 4, 4, 4, 4, 4,12,12, 8,12, 8,12,8},
        },
    };
}

```

The value of 7 using 10 bits will be added to the bitstream for the nnz value.

The coefficient values, levels and the number of encoded bits used, the length, are VLC coded. If the absolute level value is greater than 17, the number of bits used to represent the value is 28 bits. If the absolute level is less than 17 but greater than 9,

19 bits are used if 8 or less, the absolute level value is multiplied by 2 to determine the length of negative values and multiplied by 2 less 1 for original positive levels.

Listing 4.6: Part of WriteSyntaxElement_Level_VLCN function used for coefficient entropy encoding.

```

int writeSyntaxElement_Level_VLCN(SyntaxElement *se, int vlc, DataPartition *this_dataPart)
{
    int iCodeword;
    int iLength;

    int level = se->value1;    ///< coefficient value

    int levabs = abs(level);    ///< absolute coefficient value
    int sign = (level < 0 ? 1 : 0);    ///< sign of coefficient value (pos or neg)

    int shift = vlc-1;    ///< vlc is the current coefficient number
    int escape = (15<<shift)+1;

    if (levabs < escape) ///< the higher numbered coefficients are allocated less bits
    {
        int numPrefix = (levabs-1)>>shift;

        int sufmask = ~(0 xffffff)<<shift;
        int suffix = (levabs-1)&sufmask;

        iLength = numPrefix + vlc + 1;
        iCodeword = (1<<(shift+1))|(suffix<<1)|sign;
    }
    else
    {
        iLength = 28;
        iCodeword = (1<<12)|((levabs-escape)<<1)|sign;    ///< 1<<12 = 4096
    }
    se->len = iLength;
    se->inf = iCodeword;
}

```

The coded coefficient value for Table 4.7 position i0 j3, is determined by $4096 + (\text{absolute level value} - 17) \times 2 + 1$. In this case the coefficient value is now 4119, and has a length of 28 bits. This may be followed through Listing 4.6.

The next coefficient levels are also entropy coded whereby coefficient level -37 becomes 4109 using 28 bits, the third coefficient of -36 becomes 15 requiring 12 bits and level 78 becomes 26 and requires 14 bits of the bitstream.

The higher the number of a coefficient level, the less important it is to the decoding of the bitstream and the fewer number of bits allocated to it as shown in Listing 4.6.

The total zeros, run_before and coeff_token syntax elements each use their own VLC tables for entropy encoding. The coeff_token parameter maintains the total number of non-zero transform coefficient levels for individual luma and chroma coefficients.

The rest of the 4×4 luma blocks of the macroblock are entropy encoded, followed by the entropy encoding of the macroblock's chroma coefficients.

All macroblocks are now encoded until the entire slice is finished. Where FMO is not specified, macroblocks are written in a raster scan fashion, from left to right, top to bottom as shown in Figure 3.1.

Start Code Emulation Checking

The bitstream is checked to ensure that none of the data emulates the start code, prior to transmission or storage of the data. If the bitstream is not checked for extra start codes, decoding of the data may not be possible. The start code emulation checking occurs in the RBSPtoEBSP encoder function and is shown below in Listing 4.7.

Listing 4.7: RBSPtoEBSP function code for Start code emulation checking.

```

int RBSPtoEBSP(byte *streamBuffer, int begin_bytepos, int end_bytepos, int min_num_bytes)
{
    int i, j, count;

    for(i = begin_bytepos; i < end_bytepos; i++)
        NAL_Payload_buffer[i] = streamBuffer[i];

    count = 0;
    j = begin_bytepos;
    for(i = begin_bytepos; i < end_bytepos; i++)
    {
        if(count == ZEROBYTES_SHORTSTARTCODE && !(NAL_Payload_buffer[i] & 0xFC))
        {
            streamBuffer[j] = 0x03;
            j++;
            count = 0;
        }
        streamBuffer[j] = NAL_Payload_buffer[i];
        if(NAL_Payload_buffer[i] == 0x00)    ///  
checks for start code emulation
            count++;
        else
            count = 0;
        j++;
    }
    while (j < begin_bytepos+min_num_bytes) {
        streamBuffer[j] = 0x00;
        streamBuffer[j+1] = 0x00;
        streamBuffer[j+2] = 0x03;
        j += 3;
        stats->bit_use_stuffingBits[img->type]+=16;
    }
    return j;
}

```

If two zero bytes are detected in succession, after the startcode for that slice, the 0x03 emulation prevention byte is inserted into the bitstream. The bitstream count is also incremented as a result.

Reference Frames

Pictures are allocated to the data picture buffer (DPB) for storage. IDR pictures are placed into long term storage, and reference lists are updated.

As this picture is an IDR picture if the reference lists contained previous images, these would be deleted to ensure that future frames cannot reference frames prior to the IDR picture.

4.3.3 Inter Video Coding Layer Network Access Layer Unit

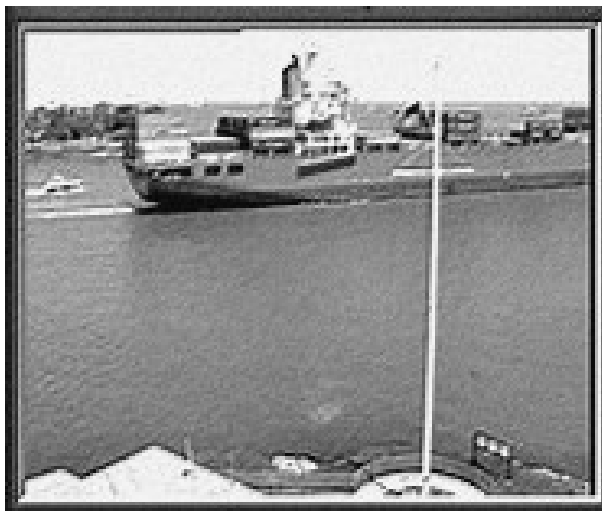


Figure 4.5: Second frame of container_qcif.yuv sourced from Kansas State University.

Figure 4.5 is the original second frame from the sampled container_qcif sequence. This frame is able to reference other frames, therefore is inter coded and is allocated to a P_SLICE image type.

Figures 4.6 and 4.7 display a tiny portion of the original video frame. The small boat appears to have actually moved by approximately 1 pixel from one frame to the next

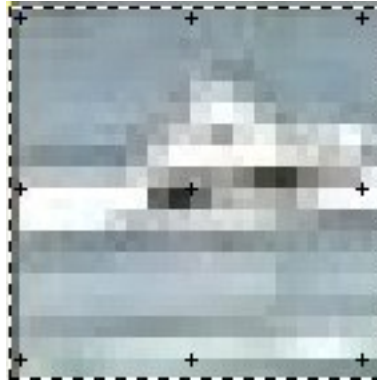


Figure 4.6: Magnified view of 2×2 block of macroblocks from the first frame of the original YUV video.

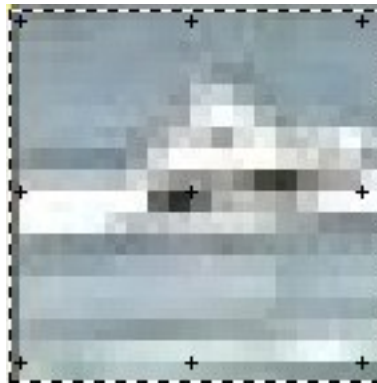


Figure 4.7: Magnified view of 2×2 block of macroblocks from the second frame of original YUV video.

frame.

A QCIF frame is 11 macroblocks wide and 9 macroblocks high. The top right hand macroblock of the magnified view in raster scan order is macroblock 35.

Table 4.10 shows luma values of the top left 4×4 block of the original macroblock, where the small boat is left most of the image.

Table 4.11 shows the luma values of the first 4×4 block of the original macroblock, where the small boat is left most of the macroblock image portion.

Motion estimation methods are now conducted, which each possible prediction method tested to define the method of least cost.

	i0	i1	i2	i3
j0	148	100	103	104
j1	164	125	78	97
j2	163	162	91	54
j3	151	160	148	79

Table 4.10: The first 4×4 luma block of macroblock 35 from frame 1.

	i0	i1	i2	i3
j0	158	109	98	105
j1	162	141	82	89
j2	159	168	114	51
j3	153	157	159	98

Table 4.11: The first 4×4 luma block of macroblock 35 from original frame 2.

DC prediction is determined from averaged values of the 4×4 blocks above and left and gives a predicted value of 137 for each of the current blocks of Frame 2, Macroblock 35.

Horizontal Prediction uses the values of the macroblock to the immediate left of the current macroblock for its prediction values. Vertical prediction uses the values of the above block as the predicted values for the current block.

Each of the following motion estimation methods are conducted in order to find the best prediction method, diagonal down left, vertical left, horizontal up, diagonal down right, vertical right and horizontal down prediction.

The prediction method that incurs the least cost will be used to derive the motion compensation differences. A motion vector will be passed to the entropy encoder, giving the direction to the prediction values.

4.4 Chapter Summary

This chapter has discussed video compression techniques used in the Recommendation. A source frame is followed from a raw video sequence through the encoding process to

a H.264 conforming bitstream.

Specific functions of the encoder source code have been referenced to show the practical implementation of H.264 video compression techniques. The encoder source code was derived from the public domain reference software made available by Dr Karsten Suehring and the Joint Video Team (2004). Extensive changes have been made in order to optimise the code for the Baseline Profile.

Chapter 5

H.264 Decoding Principles

5.1 Chapter Overview

The H.264 standard does not define the semantics of a bitstream and the methods to decode this bitstream. It is not necessary for the decoding methods defined in the standard to be exactly followed but changes must ensure the bitstream is decoded as intended.

This chapter presents a decoder block diagram and follows the decoding process with a 4×4 luma block. Specific decoding techniques are selected through the use of defined parameter sets and syntax elements.

The decoder source code listings have been supplied for major video compression techniques. The decoder source code was derived from the public domain reference software made available by Dr Karsten Suehring and the Joint Video Team (2004). The decoder software has been optimised during this research project for the Baseline Profile.

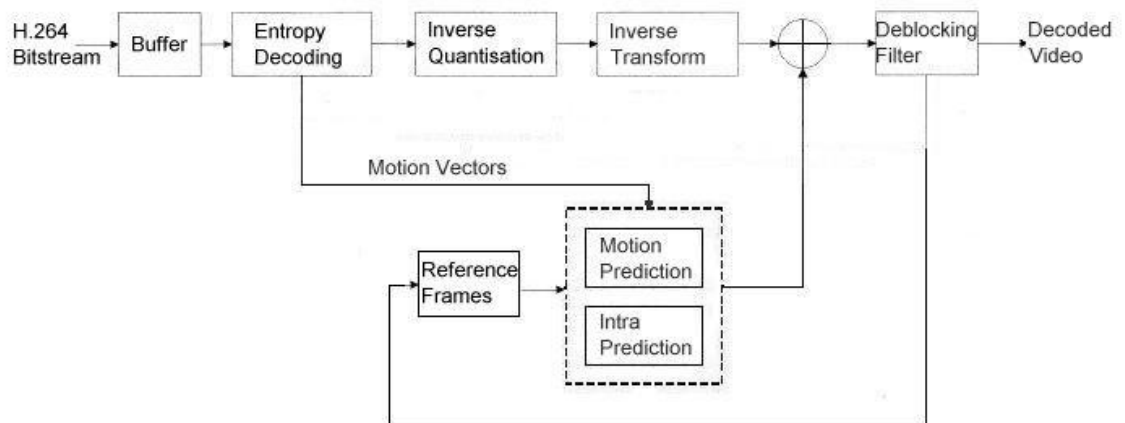


Figure 5.1: H.264 Decoder Block Diagram

5.2 The Decoder

A H.264 encoded bitstream is stored in a buffer until required for entropy decoding. The motion vectors are extracted from the bitstream and the remaining coefficients are inverse quantised and transformed. These values are added to the referenced values and pass through a deblocking filter prior to be output.

Figure 5.2 shows the decoded first frame of container_qcif sequence. The differences between this image and the original image Figure 4.2 are virtually indistinguishable.



Figure 5.2: The decoded frame of container_qcif.

5.3 The Bitstream

As with the encoder, the decoder also uses a configuration file to indicate the specific bitstream file and its associated format. The decoder screens are displayed in Figures 6.17 and 6.18.

The information supplied in the decoder.cfg file is discussed in detail in Section 6.4.

The byte stream format has been used in this chapter and is defined in Annex B of the Draft Standard (2003, pp205-7). The byte stream format specifies that each network abstract layer unit (NALU) is to be preceded by a start code and specific NALUs require a long start code.

The bitstream into the decoder is checked to locate the first start code of the first NALU. The start code of the next NALU is also located in order to supply the decoder with the NALU length.

Assuming an error free transmission or storage medium, the first NALU will be identical to the NALU defined by the encoder. Appendix C defines a sequence parameter set (SPS) NALU and if this was the first NALU of the bitstream, it would be 13 bytes long and be consistent with an expected first NALU. The type of NALU is given in the NALU header, the first byte of the NALU. Table B.2 lists the available NALU types.

5.3.1 Parameter Sets

The first NALU that will be present in a baseline profile conforming bitstream is an SPS NALU. An SPS contains decoder parameters relevant to many pictures and plays an integral role in the successful decoding process. These parameter are discussed in Appendix C.

The second NALU header expected in the bitstream is the picture parameter set (PPS) NALU and has a nal unit type of type 8. A PPS may specify appropriate parameter values required for the decoding of one or more pictures. Further information regarding PPS syntax elements and an example bitstream are located in Appendix D.

5.3.2 Intra Video Coding Layer Network Access Layer Unit

The third NALU expected to be present in the bitstream is an IDR picture. Appendix E.2 shows the partial bitstream of an encoded IDR picture from the container_qcif video sequence. The entire encoded IDR length is 9792 bytes. The NALU header of this bitstream indicates a nal type 5, being a slice without partitioning thereby conforming with the baseline profile. The nal reference idc is type 3 indicating that no use of temporal prediction, therefore this NALU contains IDR picture information.

Incoming Buffer

The slice header immediately follows the NALU header for VCL data. Slice header parameters are defined in Appendix B. A slice header is shown in byte stream format in Appendix E.

Flexible macroblock ordering (FMO) is now initiated, based upon the information supplied in the PPS. It is necessary for the decoder to use FMO information prior to decode the macroblocks, in order to output the decoded video sequence in the correct orders.

Error concealment arrays are reset to allow errors present on the bitstream to be identified.

The first macroblock's header information follows the slice header, with parameters defined in Appendix B. The macroblock header indicates the type of macroblock next present in the bitstream. Appendix E shows a macroblock header of type I4MB meaning that an intra macroblock that uses 4×4 blocks follows the header information.

The macroblock header details the chroma prediction mode, motion prediction temporal data if used, the coded block pattern which indicates the number of blocks with non zero coefficients.

The coded block pattern indicates whether there are coefficients in the current block. The block is then checked to predict the nnz, which was 0. This value is used for the

vlcnum, which lets us know which VLC table that is going to be used.

The bitstream is searched bit by bit to find a matching code from the VLC code table. Using the bitstream that was encoded in Chapter 4 a code equal to 7 with a length of 10 details that there are 4 coefficients that are not zero.

Entropy Decoding and Demultiplexing

The 4 entropy coded coefficients are removed from the bitstream.

VLC table 0 is a fixed table that is used for the first coefficient and uses the function `readSyntaxElement_Level_VLC0` to identify the next bit equal to 1 in the bitstream. This bit becomes the first bit of the data code. Using the first coefficient bits of macroblock header byte 5 in Section E.2 there are 15 zero bits prior to the first 1 being located.

The length of the numerical part of the code is determined by the number of zero bits less 3, which equals 12 bits. The next 12 bits in Section E.2 provide a value of 23. The level is found by the absolute value of $23 / 2$ plus 16. The sign bit is tested and the value is reduced by 1. The level value becomes -28. The vlc table number is also changed to 2, therefore `readSyntaxElement_Level_VLCN` shown in Listing 5.1 will be employed as the entropy decoding method for the rest of the coefficients.

The level array is updated with this value of -28, therefore `levarr[3] = -28`.

The source code listing used for entropy decoding of the remaining coefficients is the `readSyntaxElement_Level_VLCN` function shown in Listing 5.1. This function calls the `ShowBits` methods which is used to remove the desired number of bits from the bitstream for checking.

Listing 5.1: The entropy decoding `readSyntaxElement_Level_VLCN` function code.

```
int readSyntaxElement_Level_VLCN(SyntaxElement *sym, int vlc, struct datapartition *dP)
{
    Bitstream *currStream = dP->bitstream;
    int frame_bitoffset = currStream->frame_bitoffset;
    byte *buf = currStream->streamBuffer;
    int BitstreamLengthInBytes = currStream->bitstream.length;
```

```

int levabs, sign;
int len = 0;
int code, sb;

int numPrefix;
int shift = vlc-1;
int escape = (15<<shift)+1;
int addbit, offset ;

// read pre zeros
numPrefix = 0;
while (!ShowBits(buf, frame_bitoffset+numPrefix, BitstreamLengthInBytes, 1))
    numPrefix++; //basically counting the number of zeros

len = numPrefix+1; //the pre zeros of the VLC code
code = 1;

if (numPrefix < 15)
{
    levabs = (numPrefix<<shift) + 1;

    // read (vlc-1) bits -> suffix
    if (vlc-1)
    {
        sb = ShowBits(buf, frame_bitoffset+len, BitstreamLengthInBytes, vlc-1);
        code = (code << (vlc-1) )| sb;
        levabs += sb;
        len += (vlc-1);
    }

    // read 1 bit -> sign
    sign = ShowBits(buf, frame_bitoffset+len, BitstreamLengthInBytes, 1);
    code = (code << 1)| sign;
    len ++;
}
else // escape
{
    addbit = numPrefix - 15;

    sb = ShowBits(buf, frame_bitoffset+len, BitstreamLengthInBytes, (11+addbit));
    code = (code << (11+addbit) )| sb;

    len += (11+addbit);
    offset = (2048<<addbit)+escape-2048;
    levabs = sb + offset;

    // read 1 bit -> sign
    sign = ShowBits(buf, frame_bitoffset+len, BitstreamLengthInBytes, 1);
    code = (code << 1)| sign;
    len++;
}

sym->inf = (sign)?-levabs:levabs;
sym->len = len;

currStream->frame_bitoffset = frame_bitoffset+len;

return 0;
}

```

The next three incoming values of the first 4×4 block are entropy decoded from a

bitstream value of 6 using 28 bits to a coefficient of -37, value of 3 using 28 bits becomes the third coefficient of -36 and a value of 5 using 14 bits to become the fourth coefficient value of 78.

Each of these coefficients are added to the level array, with `levarr[0]` becoming 78.

When the absolute value of the level is greater than the number in the vlc table, the vlc table is changed.

The total zeros variable is searched for using the vlc table 3. The bitstream is now stepped through the designated code table, and length tables trying specific bit lengths for each code, in order to locate the next total zeros value within the bitstream.

The four run values are now retrieved from the bitstream, using the same method as for the total zeros variable. The vlc number will determine the table and specific length and code details need to be searched for bit by bit.

Inverse Scan and Quantisation

Dequantisation occurs in Listing 5.2 which is part of the `readCBPandCoeffsfromNAL` function. After the rescaling of the coefficient values they are stored in the `img->im7` array prior to transformation.

Listing 5.2: Dequantise code from `readCBPandCoeffsfromNAL` function.

```
//Dequantisation
else if (qp_per < 4)
{
    if (intra == 1)
        img->cof[i][j][i0][j0] = (levarr[k]*InvLevelScale4x4Luma_Intra[qp_rem][i0][j0]+qp_const)>>(4-qp_per);
    else
        img->cof[i][j][i0][j0] = (levarr[k]*InvLevelScale4x4Luma_Inter[qp_rem][i0][j0]+qp_const)>>(4-qp_per);
}
else
{
    if (intra == 1)
        img->cof[i][j][i0][j0] = (levarr[k]*InvLevelScale4x4Luma_Intra[qp_rem][i0][j0]<<(qp_per-4));
    else
        img->cof[i][j][i0][j0] = (levarr[k]*InvLevelScale4x4Luma_Inter[qp_rem][i0][j0]<<(qp_per-4));
}
```

Table 5.1 `levarr[0]` equals 78, `coefctr` equals 0, `runarr[0]` equals 0, `i = 0`, `j = 0`. `img->cof` matrix is shown below, following the entropy decoding and dequantisation.

	i0	i1	i2	i3
j0	4992	0	0	0
j1	-2880	0	0	0
j2	-2368	0	0	0
j3	-2240	0	0	0

Table 5.1: The first 4×4 luma block following entropy decoding and rescaling.

The coefficients of rest of the macroblock are all entropy decoded and quantised prior to inverse transformation.

Motion Compensated Prediction

Neighbours are sort for the macroblock in order to derive the prediction values. As no neighbours are found, the prediction value used will be 128.

Inverse Transform

The coefficients are transformed firstly horizontally and then vertically. Listing 5.3 shows the inverse transformation source code for the incoming coefficients.

Listing 5.3: Inverse transform portion of source code from the itrans function.

```

void itrans(struct img_par *img, ///< image parameters
            int ioff, ///< index to 4x4 block
            int joff, ///<
            int i0, ///<
            int j0, ///<
            int chroma)
{
    int i,j,i1,j1;
    int m5[4];
    int m6[4];

    Boolean lossless_qpprime = ((img->qp + img->bitdepth.luma_qp_scale)==0 && img->lossless)

    // horizontal
    for (j=0;j<BLOCK_SIZE && !lossless_qpprime;j++)
    {
        for (i=0;i<BLOCK_SIZE;i++)
        {
            m5[i]=img->cof[i0][j0][i][j];
        }
        m6[0]=(m5[0]+m5[2]);
        m6[1]=(m5[0]-m5[2]);
    }
}

```

```

m6[2]=(m5[1]>>1)-m5[3];
m6[3]=m5[1]+(m5[3]>>1);

for (i=0;i<2;i++)
{
    i1=3-i;
    img->m7[i][j]=m6[i]+m6[i1];
    img->m7[i1][j]=m6[i]-m6[i1];
}
}
// vertical
for (i=0;i<BLOCK_SIZE && !lossless_qpprime;i++)
{
    for (j=0;j<BLOCK_SIZE;j++)
        m5[j]=img->m7[i][j];

    m6[0]=(m5[0]+m5[2]);
    m6[1]=(m5[0]-m5[2]);
    m6[2]=(m5[1]>>1)-m5[3];
    m6[3]=m5[1]+(m5[3]>>1);

    for (j=0;j<2;j++)
    {
        j1=3-j;
        img->m7[i][j] = (m6[j]+m6[j1]+DQ_ROUND)>>DQ_BITS;
        img->m7[i][j1] = (m6[j]-m6[j1]+DQ_ROUND)>>DQ_BITS;
    }
}

for (i=0;i<BLOCK_SIZE && lossless_qpprime;i++)
    for (j=0;j<BLOCK_SIZE;j++)
        img->m7[i][j] = img->cof[i0][j0][i][j];
}
}

```

The 4×4 block following the inverse horizontal transform is shown below in Table 5.2.

	i0	i1	i2	i3
j0	4992	4992	4992	4992
j1	-2880	-2880	-2880	-2880
j2	-2368	-2368	-2368	-2368
j3	-2240	-2240	-2240	-2240

Table 5.2: The first 4×4 decoded luma block following horizontal inverse transform.

Table 5.3 shows the 4×4 block following the inverse vertical transform and addition of the motion prediction information.

The rest of the macroblocks are decoded in the same order.

	i0	i1	i2	i3
j0	107	107	107	107
j1	255	255	255	255
j2	231	231	231	231
j3	232	232	232	232

Table 5.3: The first 4×4 decoded luma block following the inverse vertical transform.

Deblocking Filter

The deblocking filter is used to remove the blocking artifacts induced upon a video sequence that uses a block based encoding scheme. Square image blocks are evident in a decoded frame without the filter, particularly as very few objects within an image have a square edge.

The final stage of the decoding involves applying the deblocking filter to the entire picture.

A comparison of frames with and without the deblocking filter is shown in Figure 5.2 and Figure 5.3 respectively.



Figure 5.3: The second frame of `container_qcif.264` decoded without the deblocking filter.

The deblocking filter works firstly on horizontal edges between 4×4 blocks and then on the vertical edges. When the blocks are part of differing macroblocks, stronger filtering

is employed.

5.4 Chapter Summary

The decoding process is the more common of the encoding and decoding processes. Typically a sequence will only be encoded once and will be decoded many times over. As is the case for television streaming and file transfers.

During this chapter a practical example of many of the decoding video compression techniques used by the H.264 Recommendation have been discussed including entropy decoding, inverse scanning and dequantisation, inverse transform and the deblocking filter.

Chapter 6

Graphical User Interface

6.1 Chapter Overview

The graphical user interface, GUI, provides user interaction with the H.264 standard. Specifically, the interface allows for the investigation of many video compression parameters. The GUI provides much of the input parameter error checking by limiting the user defined values.

The GUI consists of seven tabs of encoder parameters that may be defined by the user. These tabs are Main, Picture, Control, FMO, Filter, Rate Control and Miscellaneous. There are two decoder screens, detailing main and advanced parameters.

The GUI interfaces with the encoder and decoder through configuration files. The GUI was developed using Microsoft Visual Studio.Net.

6.2 Main Interface

Figure 6.1 is the opening screen of the H264 Baseline Software. This screen provides buttons to allow user interaction with the encoder and decoder parameters, as well as providing a simple explanation regarding the software's intent.

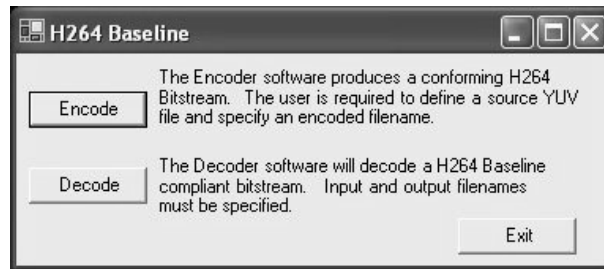


Figure 6.1: Start Screen of H264 Baseline Software

6.3 Encoder Interface

Encode Screen

Figure 6.2 displays a screen shot of the information tab. This highlights to the user that changes effected to the H264 Encoder parameters may affect the fidelity of the decoded video sequence or the encoded file size.

After changes are made to the encoder parameters, the user will be required to select the Apply button in order to allow for the parameters to be written to the encoder.cfg file in the correct sequence. This file is read by the encoder in order to provide the relevant parameters to correctly encode the video sequence. The Apply button will also initiate the encoding process.

The user may also select to cancel their interaction with the encoder.

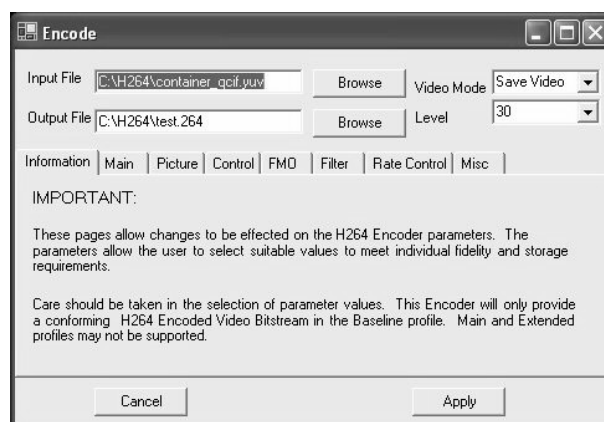


Figure 6.2: Information Screen of H264 Baseline Encode.

I have determined that the parameters at the top the Encode screen are the most important parameters for the encoder and specific file names are required to be changed by the user.

Input File This file should contain the YUV sequence to be encoded.

Output File This file will contain the encoded video sequence.

Video Mode The user may select which bitstream format the output file will employ.

Saved Video If this parameter is selected, the encoded sequence will be in the byte stream form, or as specified by Annex B. This is the default parameter.

Live Video If this parameter is selected, the encoded sequence will be in bit stream format, Good for use for RTP, such as video conferencing.

Level The user may select the required level that the encoder will use. This affects the decoded picture buffer size. There are 15 levels from which the user can choose. Level 30 is the default level.

Main Tab

The main screen provides access to the major video sequence parameters.

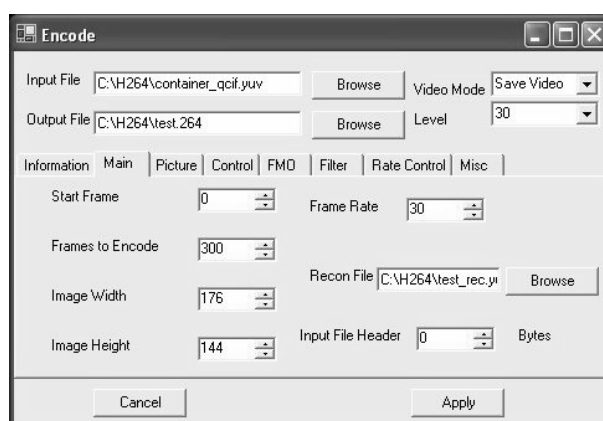


Figure 6.3: Main Screen of H264 Baseline Encode.

Start Frame This parameter allows the user to select from where in the video sequence that they would like to start encoding.

Frames to Encode This parameter allows the user to select the total number of frames that they would like to encode. I have set the number of frames to 50.

Image Width and Image Height These parameters determine the width and height of the image that the user would like to encode. As the width and height is measured in pixels, they must be divisible by 16 in order to conform to the size of macroblocks. The default is set for a QCIF size, 176×144 pixels.

Frame Rate This is the number of frames that should be displayed per second when decoded and viewed. I have used a default value of 30 frames per second

Recon File The recon file is a file that should be what is seen at the decoder. The encoder needs to know what the decoder is seeing, and therefore what predictions they will be using for motion compensation.

Input File Header This parameter is the number of bytes of the input file that is for header information. I have allowed a 0 default value.

Picture Tab

Figure 6.4 is a screen shot of the Picture screen of the H.264 Baseline Encode Software.

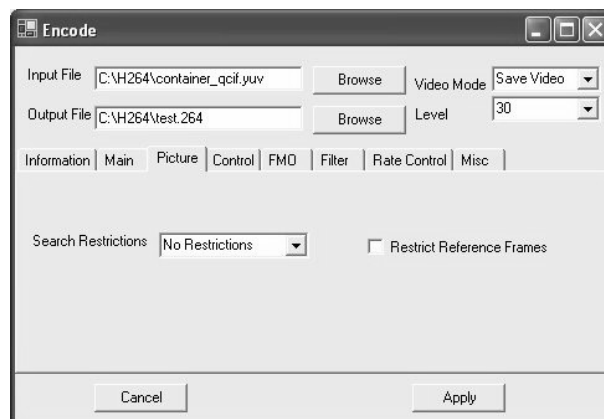


Figure 6.4: Picture Screen of H264 Baseline Encode.

The picture screen consists of only two parameters being search restrictions and reference frames restrictions.

Search restrictions allows 1 of 3 choices of searching for similar frames or blocks for

motion prediction. The search range may be restricted to using only older reference frames, both blocks and reference frames or by allowing no search restrictions to be used. I have set the Search Restrictions parameter to have a default value of No Restrictions.

Restrict Reference Frames when unchecked allows reference frames to be checked for forward prediction. The checkbox's default is to not be selected.

Control Tab

Figure 6.5 is a screen shot of the Control tab of the H.264 Baseline Encode Software. This tab page provides many control parameters that can be employed in the encoding of a video sequence.

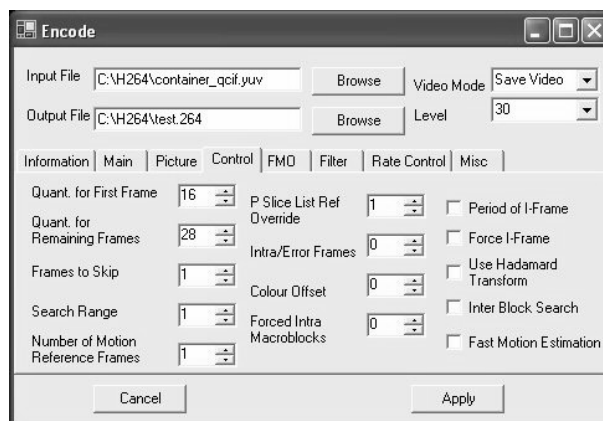


Figure 6.5: Control Tab of H264 Baseline Encode.

Each of the parameters that may be changed by the user listed on the Control Table are discussed below.

Quantisation for First Frame This variable allows the user to define the quantisation parameter that will be used in the encoding of the first frame of the video sequence. I have used a default quantisation parameter of 16.

Quantisation for Remaining Frames This parameter has a default of 28 and is used for quantisation for the remaining frames of the video sequence.

Frames to Skip This parameter allows the user to specify the number of frames to skip in between each frame to be encode. If this has a value of 1, then every second frame of the video sequence will be encoded.

Search Range The search range is the range that will be searched for motion estimation. If this value is greater than 0, then all of the surrounding blocks of a block defined by a motion vector will be searched. If this value is 0, then only the defined block will be looked at. I have allowed all of the surrounding blocks of the motion vector indicated block to be searched as default.

Number of Motion Reference Frames The number of frames allowed must be a value less than 16, and its main stipulation is that the memory requirements of the decoder picture buffer must be capable of allowing that specific number of frames. I have used a default value of 1.

P Slice List Ref Override This value determines the number of P Slices that will be allowed in the reference list and is defaulted to 1.

Intra/Error Frames This parameter is used to provide error robustness to the encoded video sequence. No extra intra blocks or frames are encoded, if this parameter is 0. If the user defines this parameter to be 1, 1 group of blocks per frame will be intra coded. If 2, 1 group of blocks will be intra coded every 2 frames. This parameter is used to prevent past macroblocks errors to be continued to future macroblocks. No extra intra blocks are to be encoded is the default.

Colour Offset This offset must be a value between -51 and 51. This parameter has a default value of 0 and changes will affect how bright the image will be when decoded.

Forced Intra Macroblocks This numerical value is used to define the minimum number of macroblocks that should be intra coded per frame.

Period of I-Frame This checkbox allows the user to define that there should be a minimum of 0 or 1 frames in between each encoded Intra Frames. I have used a default value of 0 frames.

Force I-Frame The user would check this parameter to force intra pictures to be encoded as IDR pictures. An IDR picture forces no frames to be allowed to use

any pictures prior to the IDR for motion compensation. I have left this parameter unchecked as the default setting.

Use Hadamard Transform The user would select to use Hadamard Transform when an additional transform is required.

Inter Block Search This parameter if checked allows all available block sizes to be searched.

Fast Motion Estimation If FME is checked, motion estimation will be conducted on the integer blocks, otherwise a full pel search for motion estimation is allowed. The default setting allows for a full pel search to be allowed.

FMO Tab

Figure 6.6 is a screen shot of the FMO tab of the H.264 Baseline Encode Software. This tab page allows the user to select the slice mode and whether to employ FMO.

There are four different slice modes that may be employed by the encoder. The first slice mode is off and there are therefore no restrictions placed between the macroblocks and the slice. This mode is the default slice mode. Fixed macroblocks is another slice mode, and will allow only a specific number of macroblocks to be used per slice. Fixed rate and Callback only allow slices to contain a specific maximum number of bits. Should this number be exceeded, macroblocks will be removed from the slice.

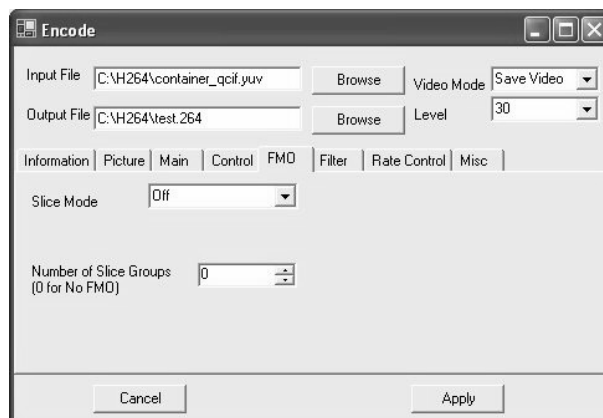


Figure 6.6: FMO Tab of H264 Baseline Encode - No FMO.

Figure 6.6 is a screen shot used where FMO is not employed. The number of slice groups allowed to reproduce the frame is 1. The macroblocks within the slice are processed in raster scan order. This tab layout shows the default values used by the software. When FMO is not employed the macroblocks are all included in one slice group and processed in raster scan order.

The six defined macroblock mapping methods are shown in Figure ???. The explicit map that is used to map individual macroblocks to slices should be contained in a text file which will be referred to as the Slice Group File Name.

Interleave Slice Map

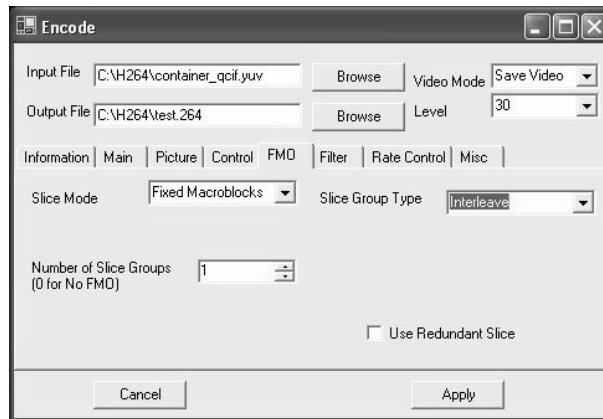


Figure 6.7: FMO Tab of H264 Baseline Encode - Interleave Slice Map.

Figure 6.7 is a screen shot used where FMO is employed with an interleave map for ordering of the macroblocks within the slice. Each macroblock row is allocated to consecutive slice groups.

Dispersed Slice Map

Figure 6.8 is a screen shot used where FMO is employed with a dispersed slice map. Each consecutive macroblock will be allocated in turn to consecutive slice groups.

Foreground with left-over Slice Map

Figure 6.9 is a screen shot of where the slice group map is foreground slice groups followed by a left over slice. A slice group map configuration file is required to specify

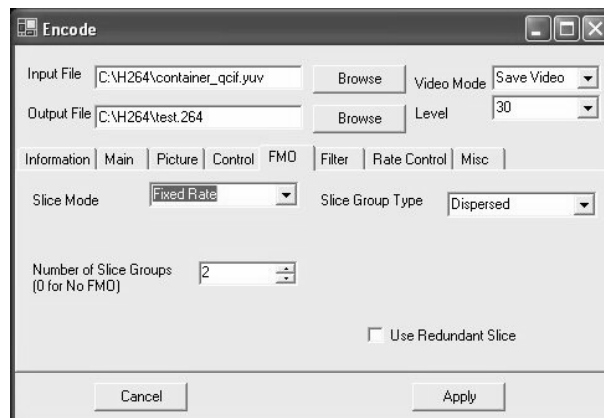


Figure 6.8: FMO Tab of H264 Baseline Encode - Dispersed Slice Map.

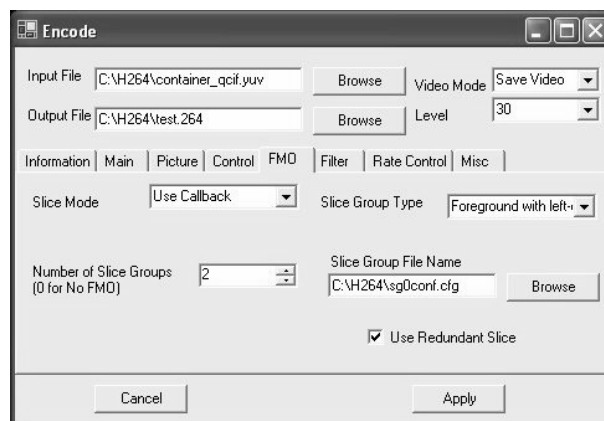


Figure 6.9: FMO Tab of H264 Baseline Encode - Foreground with left-over Slice Map.

the number of macroblocks that should be employed within each slice group.

Box-Out Slice Map

The FMO tab to use for a box-out slice map is shown in Figure 6.10. Box-out macroblock ordering may be conducted either clockwise or counterclockwise. Consecutive macroblocks are allocated to the same slice until a certain number of macroblocks is reached as specified by the user defined slice group change rate parameter.

Raster Scan Slice Map

The raster scan FMO tab is shown in Figure 6.11. Raster scan macroblock ordering is conducted from left to right, top to bottom. Macroblock ordering may also be conducted in reverse raster scan order. Each slice group will contain the number of

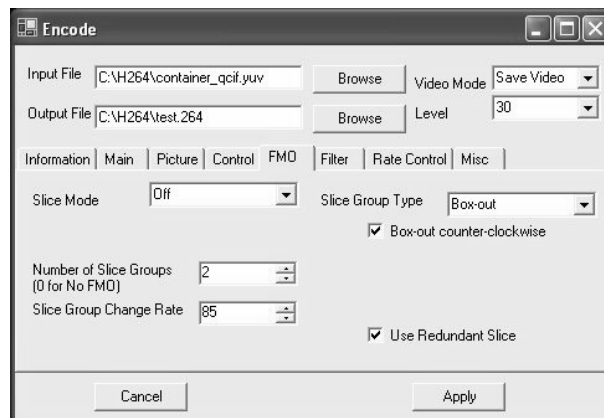


Figure 6.10: FMO Tab of H264 Baseline Encode - Box-out Slice Map.

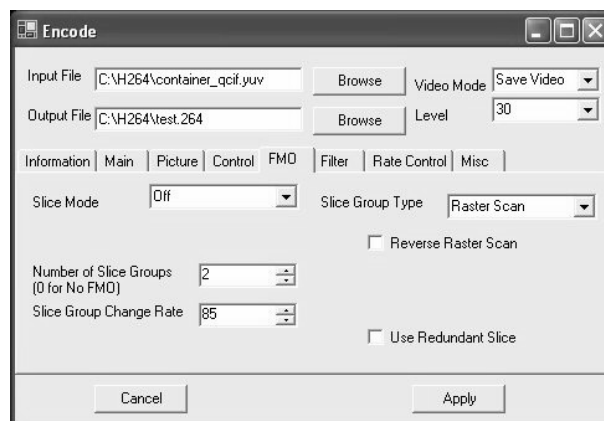


Figure 6.11: FMO Tab of H264 Baseline Encode - Raster Scan Slice Map.

macroblocks specified in the slice group change rate parameter.

Wipe Slice Map

The FMO tab shown in Figure 6.12 is used for wipe right macroblock ordering. The slice group change rate defines the number of macroblocks allocated to each slice group from top to bottom left to right. The macroblock ordering may also be conducted in the reverse order.

Explicit Slice Map

The Explicit slice map used for FMO would be completely user defined, with each individual macroblock being allocated to a specific slice group. This map would be defined in the slice group file determined by the user.

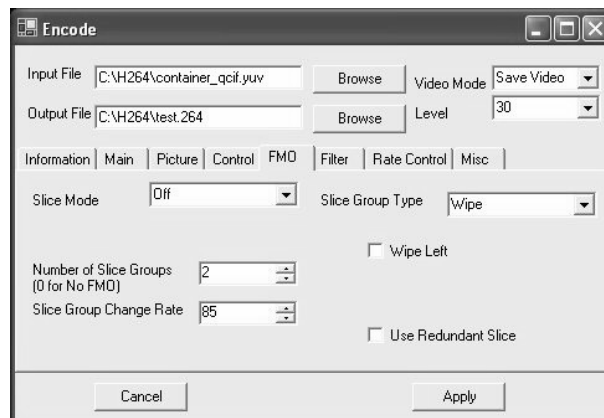


Figure 6.12: FMO Tab of H264 Baseline Encode - Wipe Slice Map.

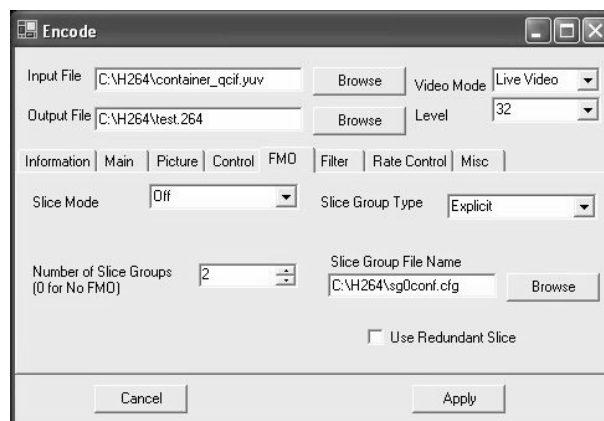


Figure 6.13: FMO Tab of H264 Baseline Encode - Explicit Slice Map.

Filter Tab

Figure 6.14 displays a screen shot of H.264 Baseline Encode Software's Filter tab.

This tab allows the Deblocking Filter to be configured or disabled.

The Alpha and Beta offsets are used in conjunction with the quantisation parameter to define when the deblocking filter will be employed. The filter is employed for small changes and will be switched off if there is a significant change in between blocks. Significant changes are expected to be as a result of a change within an image, as opposed to blocking artifacts. The larger the quantisation parameter the more block edges that will be filtered.

An example of disabling the filter completely is shown in Figure 5.3.

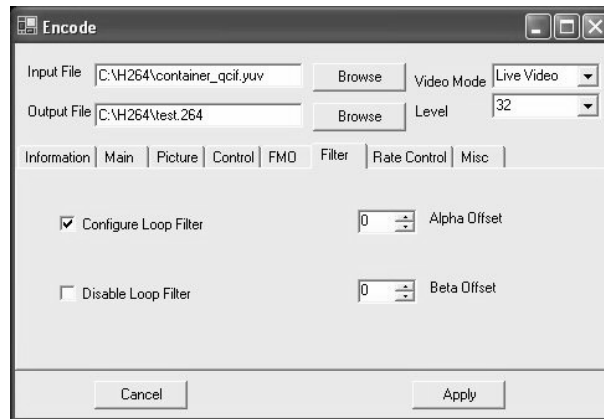


Figure 6.14: Filter Tab of H264 Baseline Encode.

The default parameters used for the loop filter is for both the Alpha and Beta offsets to be 0 and neither the loop filter configuration or disable boxes to be checked. Quantisation table 0 should be employed for both offsets when determining whether filtering should occur.

Rate Control Tab

Figure 6.15 shows the H.264 Baseline Encode Software's Rate Control tab with the preset default parameters.

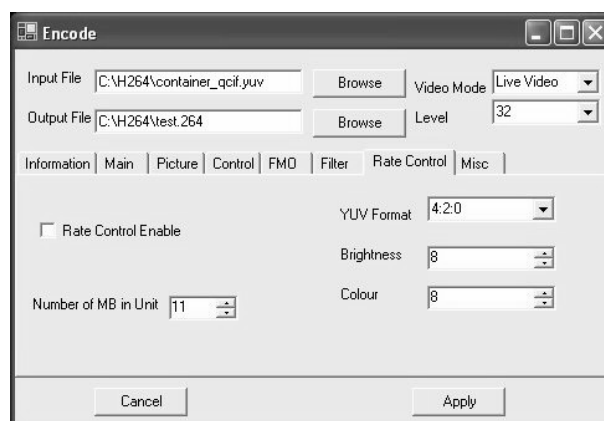


Figure 6.15: Rate Control Tab of H264 Baseline Encode.

The rate control enable check box allows for rate control to be enabled. Number of MB in unit allows the user to determine the number of macroblocks that will determine the basic unit.

The YUV format needs to be known prior to encoding a YUV video sequence to ensure that the luma and chroma components are encoded correctly. The Brightness or luma and Colour or chroma may be enhanced using these parameters, as they are used in conjunction with determining the quantisation parameter.

Misc Tab

The Miscellaneous tab allows for quantisation parameter changes to be effected part way through a sequence and for the decoder buffer to be affected. Figure 6.16 displays this screen.

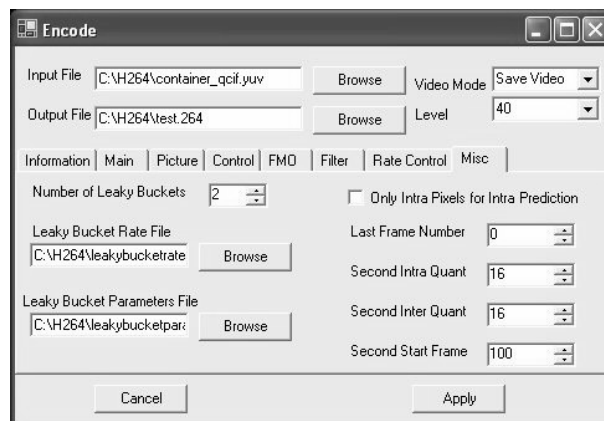


Figure 6.16: Misc Tab of H264 Baseline Encode.

A leaky bucket is a buffer used by the encoder to represent the queue between the communications channel and the decoder. The number of leaky buckets may be any value between 2 and 255. The leaky bucket parameters file will need to be defined by the user, in order to adequately utilise this parameter. The default for the number of leaky buckets is 2.

Only Intra Pixels for Intra Prediction specifies that constrained intra prediction must be used. Constrained intra prediction means that pixels encoded using intra prediction methods, may not be used for encoding inter prediction blocks. This parameter uses an unchecked box for default.

The Last Frame Number may be defined by the user to specify the last frame that should be encoded into the H264 bitstream. Leaving this parameter as 0 indicates

Frames to Encode variable will be solely used in determining the sequence length.

The second start frame provides a specific start point to allow quantisation changes to be made. The Second Intra Quant parameter specifies the quantisation parameter of the first frame of the encoding of the second video sequence. The remaining frames will use the quantisation parameter that is defined by Second Inter Quant.

6.4 Decoder Interface

Decode Screen

Figure 6.17 displays a screen shot of the main decode screen. The user is required to define the H264 compliant bitstream and the required YUV output file. Once this information is applied by the user and Apply is selected, the decoder.cfg file will be saved for accessing by the decoder executable. As with the encode configuration screens, the decoder will be initiated by the user selection of the Apply button.

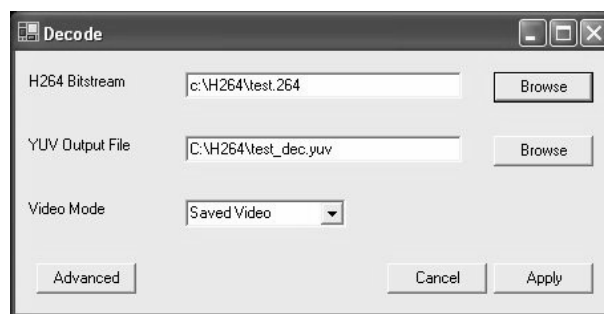


Figure 6.17: Main Screen of H264 Baseline Decode.

The user may also select the incoming bitstream's format. The saved video format uses a byte aligned format and is ideal for storage. Real time applications that require packets would employ the Live Video mode.

Advanced Decode Screen

The Advanced Decode screen is shown in Figure 6.18.

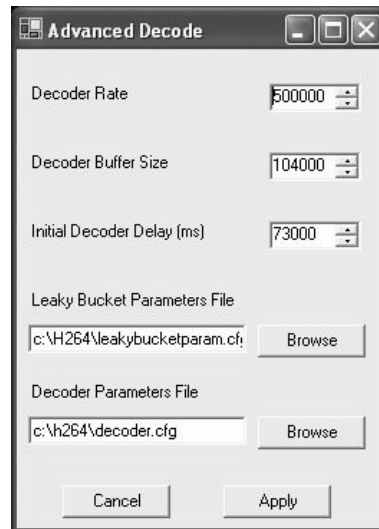


Figure 6.18: Advanced Screen of H264 Baseline Decode.

The user may define the rate, the buffer size and the delay. This has become a necessity due to the large variety of memory capabilities available, the varying bit rates and nominal delay rate. The delay rate will lower the fidelity of large pictures that take longer than the nominal time period. The leaky bucket parameters file will provide the necessary parameters to deal with the queue between the communications channel and the decoder.

6.5 Chapter Summary

The Graphical User Interface allows the user to trial many video compression techniques and the effects of altering specific parameters at the input.

The user is able to encode and decode a raw video sequence using the GUI and optimise the parameters to fulfill their fidelity and storage needs.

Chapter 7

Results

7.1 Chapter Overview

There are no definitive guides regarding the effectiveness of a video compression codec. Fidelity, complexity, encoded file size and application all play an integral role in determining the suitability of a codec.

This chapter seeks to evaluate the H.264 Recommendation in a variety of manners including parameter changing, conformance, fidelity, peak signal to noise ratio and complexity.

Parameters values are changed to gain improvements in the Recommendation's performance for fidelity, file size, complexity or data rate.

Compatibility testing is conducted against other freely available software to ensure a conforming H264 bitstream is output from the encoder and to test the decoder fidelity.

The H264 Recommendation is compared against other codecs for fidelity and file size considerations.

Peak Signal to Noise Ratio is discussed and values provided after the encoding of a H264 video sequence for both I and P frames.

7.2 Parameter Effects

Varying a parameters values can cause significant effects upon the output of a codec. Tradeoffs between the following video compression measurements are required:

- Fidelity,
- Complexity,
- File Size,
- Bit Rate.

I have used a subjective measurement for fidelity by giving a value between 0 and 10, where 0 indicates that the image is unrecognisable and 10 indicates that there is little or no difference between this image and the original.

The bit rate is a result of complexity and file size. If the processing required is too complex, the incoming bitstream may stream faster than the decoder is capable of processing the image, and the buffer may overflow. Alternatively, the encoded file may be too large for real time transmission.

I have assumed a maximum bit rate of 128kbps is available that the number of bits used to encode each frame is the same, ignoring the fact that I frames typically require more bits than P frames.

The bit rate has been determined using Equations 7.1 and 7.2. The lower bit rate value is the maximum value that is able to be employed.

$$FramesperSecond = MaxBitRate \frac{FileSize}{NumberofFrames} \times 8 \quad (7.1)$$

Max Bit Rate = 128kbps
 File Size = The size of the encoded file in bytes
 Number of Frames = Total Number of Frames encoded in file

$$ComplexityFramespersecond = \frac{1}{ComplexityRate} \quad (7.2)$$

The bit rate equations assume similar encoding and decoding times, a constant bit rate and no data errors.

Parameter	Changes		Fidelity	Complexity	File Size	Bit Rate File/Complexity
Default Values			8	90ms	184KB	4.3 / 11.1
Quantisation parameter	Pa-	QP of 16	10	100ms	453KB	1.766 / 10
Quantisation parameter	Pa-	QP of 42	1	90ms	50KB	16 / 11.1
Inter Block Search	Search all blocks		10	400ms	35KB	30.5 / 2.5
Disable Loop Filter	No	Deblocking Filter	5	60ms	184KB	4.3 / 16.7

Table 7.1: Effects of Parameter Changes.

Table 7.1 displays the effects of varying parameter values have on bit rates, complexity, file size and fidelity. The minimum bit rate value is the one that may be employed.

These parameters changes have been selected to represent the more extreme variations of changing values and that an improvement for one aspect may negatively impact another. Tradeoffs are required in order to achieve an application's individual requirements.

7.3 Conforming Bitstream and Fidelity Testing

The fidelity of an image is not easily measurable as there are many factors that may influence the quality measurement.

Richardson (2005, p21) stated that 'there are no objective measurements systems that completely reproduce the subjective experience of a man observer watching a video display.'

Fidelity opinions are therefore used in the results section.

7.3.1 Methodology

I have encoded the bitstream using the default parameters described in Chapter 6 to produce a H.264 conforming bitstream named test.264.

The H264 file is decoded using optimised H264 Baseline software, and two freely available decoders, being InterVideo's WinDVD Platinum player (2005) and ImToo Software Studio's MPEG Encoder (2005).

7.3.2 Results

Figure 4.2 shows the first frame of the original encoded sequence.

Figure 5.2 shows the first frame of the container_qcif sequence decoded using the H264 Baseline software.

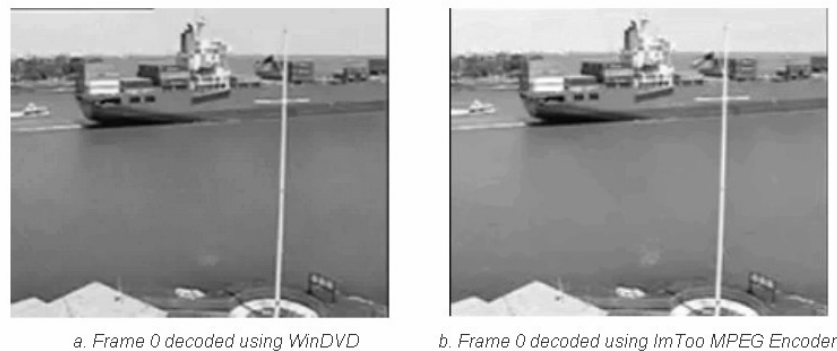


Figure 7.1: a. Decoded Frame using WinDVD Platinum. b. Decoded Frame using ImToo MPEG Encoder.

In order to use the ImToo MPEG Encoder, the file extension was required to be changed to test.h264.

Original Frame The original frame is very blocky for background areas of similar colour. The original frame is not smoothed for consistency, has sharp edges and the contrast is prominent.

Decoded Frame The decoded frame uses a contrast that is quite prominent, however

regions of a similar nature are blended together more than the original.

WinDVD Frame The decoded frame has been smoothed and also presents slightly darker than the other frames and the contrast is not as defined. The background has been smoothed to eliminate the blocky sky, but have also removed the water ripple. The flag pole detail is barely visible and would not be identified if the detail was unknown.

ImTooMPEGEncoder This image is overall lighter than the other decoded frames and has been smoothed to the point that the image appears to be slightly blurry. The water has been smoothed to remove ripple, however blocky luma variations exist within the sky and water regions. The detail is significantly reduced around the lighter coloured regions.

Commercial application of video compression techniques, generally require a tradeoff between fidelity, complexity and storage needs. Whilst the WinDVD and ImToo MPEG Encoder frames lacked detail, it was possible for the video sequences to be decoded quickly and at the correct frame rate. The optimised H.264 Baseline software whilst it provides a high fidelity video sequence, can only decode up to 5 frames per second.

7.4 Fidelity and Storage Comparison

Figure 7.2 displays the first frame of the container video sequence that has been encoded using MPEG 1, MPEG 2 and DVD formats. The image quality is very comparable with each other but the detail is significantly less than displayed in the original, see Figure 4.2 and H264 decoded frame, displayed in Figure 5.2.

Fifty frames of a raw video sequence are coded using the same video compression codecs as used for the fidelity comparisons. The resulting file sizes are listed below.

Raw YUV Video 1857KB



Figure 7.2: a. MPEG 1 b. MPEG 2 c. DVD Format.

MPEG 1 270KB

MPEG 2 288KB

DVD Format 728KB

H264 84KB

The size of the H264 file is a reduction of over 30% when compared with the MPEG formats. The data size of the H264 encoded video sequence is only 10% of the original format.

7.5 Peak Signal to Noise Ratio

The Peak Signal to Noise Ratio (PSNR) is used to describe the quality of an encoded picture when compared with the original. The differences between the encoded and original image values are used to determine this logarithmic ratio. Richardson (2005, p24) defines Equation 7.3 which allows PSNR of an image to be calculated.

$$PSNR_{dB} = 10 \log_{10} \frac{(2^n - 1)^2}{MSE} \quad (7.3)$$

n = number of bits per image sample
MSE = Mean squared error between an original and an impaired value

Richardson (2005, p24) informs that 'PSNR ratings do not necessarily correlate with true subjective quality. A human observer gives a higher importance to the face region and so is particularly sensitive to distortion in this area.' This statement is in response to a PSNR value where the background is blurred but the foreground or face maintains a high quality may indicate a lower quality picture than an image with a better PSNR value but is degraded in the wrong areas.

The PSNR values for the first frame, which is an IDR frame, of the encoded video sequence using default values are 46.34 for the Y value, 48.60 for U and 48.53 for the V component. The PSNR values for each subsequent frame is approximately 35.38 for Y, 39.9 for U and 39.7 for the V component. These values indicated that the IDR frame is a much closer representation of the original frame than of the subsequent P slices.

The first original and decoded frames are shown in Figures 4.2 and 5.2 respectively.

The difference between the PSNR values may be seen in the fidelity comparisons of Figure 4.5, the original frame versus Figure 7.3, an encoded P frame.



Figure 7.3: A decoded P Slice with an average PSNR values of 38.33dB.

The quality of the encoded images that the PSNR value represents is a measure that should be considered in codec design however subjective measurements should also be

made when determining an the image quality of particular decoders.

7.6 User Software Interaction

The readme.txt file on the accompanying cd should be read in conjunction with the installation and use of the software.

The GUI allows the software to be easily used and a variety of video compression techniques to be investigated and parameters changed. The effects of these changes may be viewed by a YUV viewer or H.264 viewing software. The developed software is expected to be used in conjunction with Chapter 6 to allow an understanding of each of the encoder and decoder parameters.

It is possible for the software to be employed without reference to this document, as minimal parameter changes are required in order to effectively encode or decode a raw video sequence.

7.7 Chapter Summary

This chapter has sought to provide a number of performance criteria from which to evaluate the H264 Recommendation. These are fidelity, file size and complexity.

Whilst the project's H264 Baseline software provides a conforming bitstream, reproduces high quality video sequences and is easy to use, further work is required to reduce encoding and decoding times when compared with other available H.264 decoders.

The changing of parameter values has shown that an improvement in one area of a codec's performance does not necessarily improve all areas.

A H264 conforming bitstream is viewed favourably when compared with other codecs due to the improved fidelity and file size. It is anticipated that increased processing power will lessen the negative complexity issues of this codec.

Chapter 8

Conclusion

This research project investigated and implemented the International Telecommunication Union's H.264 Recommendation also known as the International Standards Organisation's MPEG-4 / Advanced Video Codec.

The H.264 Recommendation provides an interoperable global standard for effective and reliable video compression. The standard uses advanced video coding techniques, with effort made to encompass a broad application base to ensure it is well postured to be widely employed internationally.

8.1 Achievement of Objectives

The objectives of the research project are contained in the Project Specification in Appendix A.

All of the objectives were met with the exception of creation and evaluation of software for real time applications and mobile solutions.

This dissertation combines H.264 video compression techniques with the relevant source code to allow the display and the implementation of encoding and decoding algorithms.

Changes to the reference software were conducted in strict consultation with the defined

Recommendation. This was in order to ensure a compliant bitstream availability at the output of the encoder and that a conforming bitstream at the input to the decoder would be treated appropriately.

The graphical user interface provides easy operation of the encoder and decoder and allows the user to define their own video compression requirements.

The fidelity of an encoded video sequence is subjectively compared against other codecs, H.264 decoders and with respect to parameter changes. The effects on bit rate changes have been simulated and are directly affected by the complexity of the bitstream and the tradeoff required with fidelity.

8.2 Future Work

As the ITU-T Recommendation was formally released in May 2003 and commercial decoders are slowly being released onto the market. H.264 encoded video sequences are not currently commonly available, however the standard is anticipated to be employed by major video organisations in the near future.

There is a considerable amount of future work that can be conducted. Specifically for real time applications of the Recommendation as detailed in objective 6 of Appendix A.

The processing speed of the encoder and decoder used in this project needs to be improved to enable its use for real time and streaming applications.

Video compression algorithms are continually being developed and with increased processing power more complex techniques are able to be employed.

List of References

- Ajay K. Luthra, G. J. S. & Wiegand, T. (2003), 'Introduction to the Special Issue on the H.264/AVC Video Coding Standard', *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 13, No. 7, July 2003 .
- Atul Puri, X. C. & Luthra, A. (2004), 'Video coding using the H.264/Avc compression standard', *Signal Processing: Image Communication*, v 19, n 9 SPEC. ISS., October, 2004, p 793-849 .
- Container.qcif* (2005), *Kansas State University* .
- Hancock, D. N. (2005), *Research Project Project Reference Book*, Distance Education Centre, USQ, Toowoomba, Australia.
- Henrique S. Malvar, Antti Hallapuro, M. K. & Kerofsky, L. (n.d.), 'Low Complexity Transform and Quantization in H.264/AVC journal ='
- InterVideo (2005), 'Windvd platinum software', *InterVideo Website* .
- Jordi Ribas-Corbera, P. A. C. & Regunathan, S. L. (2003), 'A Generalized Hypothetical Reference Decoder H.264/AVC', *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 13, No. 7, July 2003 .
- Jorn Ostermann, Jan Bormans, P. L. D. M. M. N. F. P. T. S. & Wedi, T. (2004), 'Video coding with H.264/Avc: Tools, Performance, and Complexity', *IEEE Circuits and Systems Magazine*, v 4, n 1, First Quarter, 2004, p 7-28 .
- Leis, J. (2003), *Digital Signal Processing - A MATLAB-Based Tutorial Approach*, Research Studies Press Ltd.

- Michael Horowitz, Anthony Joch, F. K. & Hallapuro, A. (2003), ‘H.264/AVC Baseline Profile Decoder Complexity Analysis’, *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 13, No. 7, July 2003 .
- Organisation, I. S. (2004), *Part 15: Advanced video coding (AVC) file format AS ISO/IEC 14496.10*, Standards Australia.
- Peter List, Anthony Joch, J. L. G. B. & Karczewicz, M. (2003), ‘Adaptive Deblocking Filter’, *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 13, No. 7, July 2003 .
- Ralf Schafer, T. W. & Schwarz, H. (2003), ‘The Emerging H.264/AVC Standard’, *EBU Technical Review - January 2003* .
- Richardson, I. E. G. (2003), *H.264 and MPEG-4 Video Compression: Video Coding for Next-generation Multimedia*, John Wiley & Sons.
- Studio, I. S. (2005), ‘Imtoo mpeg encoder software’, *ImToo Website* .
- Suehring, D. K. & Team, J. V. (2004), ‘Joint model 9.0 software and documentation’, *Image Processing Internet Website* .
- Sullivan, G. J. & Wiegand, T. (2005), ‘Video Compression — From Concepts to the H.264/AVC Standard’, *Proceedings of the IEEE*, Vol. 93, No. 1, January 2005 .
- Tanenbaum, A. S. (2003), *Computer Networks*, Pearson Education International.
- Thomas Wiegand, G. J. S. & Luthra, A. K. (2003a), *Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification ITU-T Rec. H.264 / ISOIEC 14496-10 AVC*, ITU-T & ISO/IEC.
- Thomas Wiegand, Gary J. Sullivan, G. B. & Luthra, A. K. (2003b), ‘Overview of the H.264/Avc Video Coding Standard’, *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 13, No. 7, pp 620-636, July 2003 .
- Ville Lappalainen, A. H. & Hamalainen, T. D. (2003), ‘Complexity of Optimized H.26L Video Decoder Implementation’, *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 13, No. 7, July 2003 .

Wedi, T. & Musmann, H. G. (2003), 'Motion and Aliasing-Compensated Prediction for Hybrid Video Coding', *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 13, No. 7, July 2003 .

Wien, M. (2003), 'Variable Block-Size Transforms for H.264/AVC', *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 13, No. 7, July 2003 .

Appendix A

Project Specification

University of Southern Queensland
FACULTY OF ENGINEERING AND SURVEYING

ENG 4111/2 Research Project

PROJECT SPECIFICATION

FOR: **Belinda FARMER**
TOPIC: Video Compression using ITU-T Recommendation H.264.
SUPERVISOR: Wei Xiang
ENROLMENT: ENG 4111 – S1, X, 2005;
ENG 4112 – S2, X, 2005.

PROJECT AIM: This project aims to investigate the H.264 standard and implement the video compression algorithms using low bandwidth solutions. The scope of the project also includes evaluating the developed software using varying fidelity and bit rates for both real time and delayed applications.


PROGRAMME: **Issue B, 25 March 2005**

1. Research the background information relating to the ITU-T Recommendation H.264/AVC.
2. Develop a program, written in C/C++, utilising available H.264 Reference Software as a basis for the optimized project code.
3. Provide a graphical user interface for the video compression software to enable its use by a variety of users.
4. Evaluate the software using varying fidelity and bit rates.
5. Evaluate the software for both real time and delayed applications.

As time permits

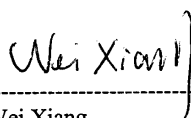
6. Minimise the software for use with mobile solutions, whereby processor speeds and memory availability are considerations.

AGREED:



Belinda Farmer

Dated 25/3/05



Wei Xiang
7/4/2005

Appendix B

NAL and Picture Parameters

B.1 Chapter Overview

The chapter includes the parameters that are required to make up a picture, including slice and macroblock semantics.

Network access layer (NAL) parameters, nal unit types and header information is also defined to enable the quick recognition of NALU header information from the bitstream.

CAVLC entropy encoding parameters are also defined later in this chapter.

B.2 Network Access Layer Parameters

The network access layer (NAL) is specified to allow the successful transmission or storage of video data. A network access layer unit (NALU) is the individual unit that is used to provide the generic formatting for the data.

NALUs can primarily be divided into 2 types, video coding layer (VCL) and non-VCL. Table B.2 displays the different kinds of data that make up each of these types.

A NALU consists of specific header information, which includes what type of NALU it

is. This allows the decoder to determine the requirements of the incoming bitstream.

The Recommendation's byte stream format requires that preceding the NALU header information, a start code and extra padding bytes be included to distinguish between subsequent NALUs. This demarcation method is not required for NALUs for packet-oriented transport.

A Network Access Layer Unit has a one byte header that consists of the following parameters:

forbidden_zero_bit This parameter is hardcoded to 0.

nal_ref_idc If this parameter is 0, it indicates that the slice or slice data partition contained with the NAL unit is part of a non-reference picture, or of another specified nal_unit_type.

nal_unit_type This parameter specifies the type of RBSP data structure contained in the NALU unit.

This 8 bit header is constructed as follows:

Bit 7 - MSB forbidden_zero_bit

Bit 6 - nal_reference_idc MSB

Bit 5 - nal_reference_idc LSB

Bit 4 - nal_unit_type MSB

Bit 3 - nal_unit_type

Bit 2 - nal_unit_type

Bit 1 - nal_unit_type

Bit 0 - nal_unit_type LSB

Table B.2 combines the various NALU parameters into the one table for easy reference when attempting to determine the bitstream.

nal_unit_type	Content of NAL unit	VCL/non-VCL	nal_ref_idc
0	Unspecified - may be used by any application	-	-
1	Coded slice of a non-IDR picture	VCL	≠0
2	Coded slice data partition A	VCL	≠0
3	Coded slice data partition B	VCL	≠0
4	Coded slice data partition C	VCL	≠0
5	Coded slice IDR picture	VCL	≠0
6	Supplemental enhancement information SEI	non-VCL	0
7	Sequence Parameter Set	non-VCL	≠0
8	Picture Parameter Set	non-VCL	≠0
9	Access unit delimiter	non-VCL	0
10	End of sequence	non-VCL	0
11	End of stream	non-VCL	0
12	Filler Data	non-VCL	0
13-23	Reserved - Decoder will remove NALU from bitstream and discard	-	-
24-31	Unspecified - may be used by any application	-	-

B.3 Slice Parameters

The parameters as described in the Recommendation are:

first_mb_in_slice This parameter provides the address of the first macroblock in the slice. As arbitrary slice order is allowed in the baseline profile, this value does not have to be in numerical order for other slices of the same picture.

slice_type This parameter indicates whether the slice is an I or P slice. The value of 0 or 5 indicates a P slice, whereas the value of 2 or 7 indicates an I slice. Other slices, not conforming to the baseline profile, use other values between 0 and 9.

pic_parameter_set_id This parameter specifies the picture parameter set, PPS, that is in use. The value for this parameter will always be 0 when using this encoder.

frame_num This value is used to identify the short term reference frame. If the picture is an IDR picture, this value shall be set to 0, otherwise may be any value up to an including the number of frames to be encoded.

field_pic_flag This syntax element is not required to be used in the bitstream, as macroblock adaptive frame/field mode coding is used, mbaff, is not allowed in the baseline profile.

bottom_field_flag This syntax element is also not required, as it refers to the bottom coded field of an mbaff used slice.

idr_pic_id This syntax element identifies an IDR picture, the value is hardcoded to 0 in the encoder software.

pic_order_cnt_lsb This parameter specifies the picture order count modulo MaxPicOrderCntLsb for the top field of a coded frame.

delta_pic_order_cnt_bottom This parameter specifies the picture order count, POC, difference between the top and bottom fields of a coded frame, and when it is not present in the bitstream, it is inferred to be 0. This parameter is not used in the slice header for this encoder.

delta_pic_order_cnt[0] This parameter is used for the POC difference from the expected count for the top field. This parameter is only required for POC mode 1, and when this parameter is not present, it is inferred to be 0.

delta_pic_order_cnt[1] This parameter is used for the POC difference from the expected count for the bottom field. It is also only required for POC mode 1, and if the parameter is not present, it is inferred to be 0.

redundant_pic_cnt This parameter is equal to 0 for slices belonging to the primary coded picture. For a redundant coded picture this count shall be greater than 0.

When this parameter is not present in the bitstream, it is inferred to be 0.

num_ref_idx_active_override_flag This parameter is included in the bitstream for P slices and will always be 0, because MBAFF coding is not allowed in the baseline profile, therefore parameter `num_ref_idx_l0_active_minus1` will always be between 0 and 15 inclusive.

ref_pic_list_reordering_flag_l0 If this flag equals 1, it indicates that `reordering_of_pic_nums_idc` is present for reference picture list 0.

reordering_of_pic_nums_idc This parameter specifies which of the reference pictures will be remapped.

reordering_of_pic_nums_idc = 0 `abs_diff_pic_num_minus1` is present and therefore subtract this value from a picture number prediction value.

reordering_of_pic_nums_idc = 1 `abs_diff_pic_num_minus1` is present and therefore add this value to a picture number prediction value.

reordering_of_pic_nums_idc = 2 `long_term_pic_num` is present and long-term reference picture number is defined.

reordering_of_pic_nums_idc = 3 This value signifies the end loop for reordering.

abs_diff_pic_num_minus1 This value gives the absolute difference between the picture numbers for reordering.

long_term_pic_num This value specifies the long term picture number to be moved.

no_output_of_prior_pics_flag This syntax element specifies how the decoded picture buffer, DPB, treats previously coded pictures.

long_term_reference_flag This parameter when 1 ensures that the current IDR picture is used for long-term reference.

adaptive_ref_pic_marking_mode_flag This parameter is used for non-intra slices, and when it is equal to 0, it uses a sliding window reference picture marking mode, and when 1, an adaptive reference picture marking mode is used, as well as the additional following parameters.

memory_management_control_operations There are six memory management control operations, being to end the loop, marking a short term picture unused for reference, marking a long term picture as unused for reference, assigning a long term frame index to a short term picture, specifying the maximum long term frame index, marking all reference pictures as unused and assigning a long term frame index to the current decoded picture. The following additional parameters are further included depending upon the memory management control operation used. `Difference_of_pic_nums_minus1` is used to mark a long term frame index or short term reference picture as unused. `Long_term_pic_num` is used to mark a long term reference picture as unused. `Long_term_frame_idx` is used to assign a long term frame index to a picture. `Max_long_term_frame_idx_plus1` is used to specify the maximum value of the index allowed for long term reference pictures.

slice_qp_delta This quantisation parameter is used for the user defined quantisation parameter minus 26.

disable_deblocking_filter_idc This parameter specifies the operation of the deblocking filter, whether it should be disabled, and for what edges. If this parameter is 0 or 2, the following parameters are required.

slice_alpha_c0_offset_div2 This parameter specifies the offset used in accessing the a and t deblocking filter tables.

slice_beta_offset_div2 This parameter specifies the offset to use in accessing the b deblocking filter tables.

slice_group_change_cycle This parameter is used to derive the number of slice group map units to use in slice group 0 for the box-out, wipe or raster scan mapping methods.

B.4 Macroblock Parameters

The parameters as described in the Recommendation are:

mb_type This parameter specifies the macroblock type. For I slices the types available are Intra_4x4 prediction and Intra_16x16 prediction. For P slices, macroblock types include one luma 16x16 partition and chroma samples, two luma 16x8 or 8x16 partitions and chroma samples, using a sub-macroblock and a final macroblock type that indicates that there is no further data present for the macroblock in the bitstream.

prev_intra_4x4_pred_mode_flag This parameter specifies the intra_4x4 prediction of the indicated 4x4 luma block.

intra_chroma_pred_mode This parameter specifies the type of spatial prediction is to be used for chroma, following the luma macroblock intra coding. The modes are:

mode 0 DC prediction.

mode 1 Horizontal prediction.

mode 2 Vertical prediction.

mode 3 Plane prediction.

coded_block_pattern This parameter specifies which of the six 8x8 luma and chroma blocks contain non-zero transform coefficient levels.

mb_qp_delta This parameter provides the difference between the luma quantisation parameter of the previous macroblock and the luma quantisation parameter of the current macroblock.

B.5 CAVLC Parameters

coeff_token This value represents the total number of non-zero transform coefficient levels and the number of trailing one transform coefficient levels in a level scan.

trailing_ones_sign_flag This indicates that the corresponding transform coefficient level should be decoded as 1.

level_prefix This parameter specifies the value of a non-zero transform coefficient level.

level_suffix This parameter is used in conjunction with the `level_prefix` to specify the value of a non-zero transform coefficient level.

total_zeros This parameter specifies the total number of zero-valued transform coefficient levels.

run_before The parameter is for the number of consecutive transform coefficient levels in the scan with a zero value.

Appendix C

Sequence Parameter Set

C.1 Chapter Overview

A sequence parameter set is a set of syntax elements used to describe the necessary common encoding information for a complete video sequence.

C.2 Parameters

The parameters as described in the Recommendation are:

profile_idc This parameter indicates which profile the bitstream conforms to. The encoder in this research project conforms to the baseline profile, which is designated by the value 66.

level_idc This parameter indicates the level that the bitstream conforms to. The meaning of the levels has been described in table .

constraint_set0_flag This parameter indicates that the bitstream obeys all constraints specified by the Baseline profile, when the flag is equal to 1. When the flag is equal to 0 the bitstream may or may not obey the baseline profile constraints.

constraint_set1_flag This parameter is the same as the above flag, but with respect to the H.264 Main profile.

constraint_set2_flag This parameter is the same as the above flag, but with respect to the H.264 Extended profile.

constraint_set3_flag This parameter is not defined in my copy of the Recommendation, but I suggest that it is probably with respect to the FREXT parameters.

reserved_zero_4bits This parameter is simply hardcoded to 0, and is ignored by the decoder.

seq_parameter_set_id The encoder used in this research project currently has the id hardcoded to 0, however it can be any value between 0 and 31, inclusive. The Recommendation indicates that the seq_parameter_set_id should be changed when the values of different syntax elements are used, as opposed to actually changing the values of the syntax elements themselves.

log2_max_frame_num_minus4 This parameter is a value between 0 and 12 inclusive. It is derived by determining the minimum number of bits to represent the number of frames to decode as retrieved from encoder.cfg, and then minus 4 for this number of bits required. If this value is less than 0, the value will become 0. Therefore the maximum number of frames that can be encoded is $2^{16} = 65536$.

pic_order_cnt_type This parameter determines whether POC mode 0, 1, or 2 will be implemented to decode the picture order count. A picture order count is used for decoder conformance checking.

POC mode 0 pic_order_cnt_type is 0, then the SPS parameter log2_max_pic_order_cnt_lsb_minus4 is used, which specifies the value of the variable MaxPicOrderCntLSB.

log2_max_pic_order_cnt_lsb_minus4 This parameter is a value between 0 and 12 inclusive, it is derived by determining the minimum number of bits to represent the twice the number of frames to encode multiplied by the number of frames to skip plus 1 as part of the input parameters required of the encoder. Once the number of bits is determined, four is taken from this value.

POC mode 1 delta_pic_order_always_zero_flag If this flag is 0 it indicates that `delta_pic_order_cnt[0]` is present in the slice header and `delta_pic_order_cnt[1]` may be present in the slice header of the sequence.

offset_for_non_ref_pic This parameter is used to calculate the value of the POC of a non-reference picture.

offset_for_top_to_bottom_field This parameter used to calculate the the POC of the bottom field of a frame.

num_ref_frames_in_pic_order_cnt_cycle This parameter is used in the decoding process for picture order count and is in the range of 0 to 255, inclusive.

offset_for_ref_frame[] This parameter is an element of a list of `num_ref_frames_in_pic_order_cnt_cycle`.

POC mode 2

num_ref_frames This parameters specifies the total number of short and long term reference frames used for inter prediction of any picture in the sequence. This value is in the range of 0 to 16 inclusive and is input by the user.

gaps_in_frame_num_value_allowed_flag This flag specifies the values of `frame_num` are allowed. If the flag is 0 and the `frame_num` is not equal to the `PrevRefFrameNum`, then `frame_num` needs to be equal to $(\text{PrevRefFrameNum} + 1) \% \text{MaxFrameNum}$. If `frame_num` does not equal to either of these values, then the flag which change to 1, to indicate the unintentional loss of pictures, and to begin a decoding process to deal with these gaps.

pic_width_in_mbs_minus1 This parameter specifies the width of each picture in units of macroblocks minus 1. This value is determined by the picture width in pels, divided by 16, less 1.

pic_height_in_map_units_minus1 This parameter specifies the height in slice group map units of a decoded frame minus 1. This value is calculated from the picture height in pels, divided by 16, less 1.

frame_mbs_only_flag In order to conform to the baseline profile, this flag is hard-coded to 1, which indicates that all of the pictures of this video sequence are frames coded that contain only frame macroblocks.

mb_adaptive_frame_field_flag This flag when 0 specifies that there is no switching between frame and field macroblocks. This flag is inferred to be 0 when it is not present, and must be 0 in order to conform with the baseline profile, it is not included in the SPS of this project's encoder.

direct_8x8_inference_flag This flag is used only for B slices, and therefore is not required for the baseline profile. It has been hardcoded to 0.

frame_cropping_flag When this flag is 1, it indicates that the frame cropping offset parameters are next in the SPS, and when 0 specifies that the parameters are not present.

frame_crop_left_offset

frame_crop_right_offset

frame_crop_top_offset

frame_crop_bottom_offset These parameters specify the horizontal and vertical coordinates of a cropping rectangle, within which to encode the samples of the frame.

vui_parameters_present_flag This flag when 1 specifies that the vui_parameters are present next in the bitstream, and when 0, indicates that they are not present.

C.3 Bitstream information

The SPS is first created as a raw byte sequence payload.

A new data partition is created for the parameter set and contains the UVLC processing function and a Bitstream data structure.

A Syntax Element for Profile IDC is parsed using the `u_v()` function, where 8 bits will be assigned to Profile.IDC. The function is defined in the Recommendation as `u(n)`, where `n` refers to the number of bits that this syntax element will use. The parsing process writes a binary representation of the parameters unsigned integer value starting at the most significant bit (MSB) of the byte first. The syntax Element is of type SE_HEADER.

The next syntax element which is `constrained_set0_flag` is set to 1 indicating that the bitstream will conform to the Baseline profile. This syntax element uses the `u_1()` function, which means that the syntax element uses only 1 bit and is parsed using the `u(n)` function.

The next three syntax elements are `constrained_set1_flag`, `constrained_set2_flag` and `constrained_set3_flag`. Each of these flags are set to 0 and parsed using the `u_1()` function.

`Reserved_zero_4bits` is set to 0 by bitstreams that conform to this Recommendation, therefore the value of this bit is zero. The decoder ignores the value of this parameter. This syntax element is parsed using `u_v()` and uses 4 bits to represent its 0 value.

The next syntax element is `level_idc` which is a value defined in the Graphical User Interface (GUI). I have set a default value of this parameter to 30. This Syntax element is parsed using `u_v()`, and uses 8 bits.

The next syntax element is `seq_paramater_set_id` and is currently hardcoded to 0. This is parsed using `ue_v()` which means that it is an unsigned integer Exp-Golomb-coded syntax element with the left bit first. An unsigned Exp-Golomb coded integer is given by the actual parameter's value + 1.

The parsing process for these Syntax Elements is designated in the Recommendation as `ue(v)`. The value of the parameter is first converted to an Exp-Golomb bit string.

The parsing process for decoding Exp-Golomb codes begins with reading all of the bits in the bitstream up to and including the first non-zero bit. When decoding the bitstream, the number of leading bits that are equal to 0 are counted and shown in Equation C.1 as the variable Number of Leading Zero Bits. The Number of Leading Zero Bits is determined by the number of zeros at the current location in the bitstream up to and including the first non-zero bit. The resulting variable termed `codeNum` in the Recommendation may be determined by the following equation:

$$codeNum = 2^{NumberOfLeadingZeroBits} - 1 + (therestoftheExp-Golombbitstring) \quad (C.1)$$

The syntax element `seq_paramater_set_id` uses a 0 value, therefore the Exp-Golomb coded bit string will be a 1.

The next SPS parameters to be parsed are the `log2_max_frame_num_minus4` and the `pic_order_cnt_type` variable which use `ue_v()`.

The next SPS parameter is `log2_max_pic_order_cnt_lsb_minus4`. I have used the value of 7 for this parameter which was derived by using 300 for the number of frames to be encoded and by skipping every second frame. $2 \times 300 \times 2 = 1200$ or 100 1011 0000 and by reducing the number of bits required (11) by 4 gives the parameter a value of 7.

If the POC mode was 1, the following parameters would also need to be set.

- `delta_pic_order_always_zero_flag`
- `offset_for_non_ref_pic`
- `offset_for_top_to_bottom_field`
- `num_ref_frames_in_pic_order_cnt_cycle`
- `offset_for_ref_frame` for each of the ref frames in the above parameter.

The next SPS parameter required for the bitstream is `num_ref_frames` and is parsed using `ue_v()`.

`Gaps_in_frame_num_value_allowed_flag` parsed using `u_1()` is the next SPS parameter to be included in the bitstream.

The parameter `pic_width_in_mbs_minus1` specifies the width of each decoded picture minus 1 in units of macroblocks and this syntax element is parsed using `ue_v()`.

The next SPS parameter for the bitstream is `pic_height_in_map_units_minus1` and specifies the height minus 2 in slice group map units of a decoded frame or field.

The next SPS parameter is `frame_mbs_only_flag`. In order to conform to the baseline profile, this flag must be equal to 1, which specifies that every coded picture of the

coded video sequence is a coded frame containing only frame macroblocks. The value of this parameter is therefore 1, and it is parsed using `u_1()`.

The SPS parameter `direct_8x8_inference_flag` is used for B slices. It is not required for the baseline profile and has been hardcoded to 0. This flag is parsed using function `u_1()`.

The next SPS parameter is the `frame_cropping_flag` and I have used a value of 0, which indicates that the frame cropping offset parameters are not present. This flag is also parsed using `u_1()`.

The next SPS parameter is the `vui_parameters_present_flag` which indicates the the `vui_parameters` structure is not present next in the bitstream and is the last parameter of the SPS.

The unfinished string of data bits (`sodb`) of the SPS is completed to the 8th byte. A 1 is added as the lsb and the rest of the byte is complete in 0s.

The `rbsp` is converted to a `nal` using the Annex B byte stream format. A SPS NALU requires a long start code which means that the start code prefix length is be 4 bytes long.

The Annex B byte stream format provides the start code demarcation between NALU's.

The first byte of the `nal` buffer is filled with `nal` header information. The 7th bit is 0, which is the hardcoded value of the `nalu_forbidden_bit`. The 5th bit becomes the `nal_reference_idc` value of 3. Finally the 5 lsbs are used to represent the `nal_unit_type`, which is type 7, indicating it is a `seq_parameter_set_rbsp`.

There have therefore been 104 bits written for the SPS NALU, including the start code, the NALU header and the SPS. A completed sample bitstream is shown in the section below.

C.4 SPS NALU Bitstream

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

Long Start Code - 4 bytes

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1

SPS NALU Header

0	1	1	0	0	1	1	1
---	---	---	---	---	---	---	---

SPS Byte 1 \bar{B} (66)

1	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

SPS Byte 2 = (128)

1	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

SPS Byte 3 = (30)

0	0	0	1	1	1	1	0
---	---	---	---	---	---	---	---

SPS Byte 4 = (154)

1	0	0	1	1	0	1	0
---	---	---	---	---	---	---	---

SPS Byte 5 \bar{I} (33)

0	0	1	0	0	0	0	1
---	---	---	---	---	---	---	---

SPS Byte 6 \bar{J} (5)

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

SPS Byte 7 \bar{O} (137)

1	0	0	0	1	0	0	1
---	---	---	---	---	---	---	---

SPS Byte 8 = (136)

1	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

Appendix D

Picture Parameter Set Bitstream

D.1 Chapter Overview

A picture parameter set is a set of syntax elements used to describe the necessary common encoding information for a picture sequence.

The Network Access Layer that immediately follows the Sequence Parameter Set is the Picture Parameter Set.

The Picture Parameter Set, PPS, is the set of parameters that is used for pictures in the coded video sequence. This parameter set allows the encoder to designate particular methods of coding the video sequence as well as the use of flexible macroblock ordering, FMO.

D.2 Parameters

The parameters as described in the Recommendation are:

pic_parameter_set_id This parameter is used in the slice header, to identify the picture parameter set associated with it. In the software of this research project, all `pic_parameter_set_id` is set to 0, however the Recommendation allows any value

in the range of 0 to 255.

seq_parameter_set_id This parameter is used to refer to the active sequence parameter set and the Recommendation allows any value in the range of 0 to 31, however in the software of this research project this value is set to 0.

entropy_coding_mode_flag This parameter is used to indicate the method of entropy encoding to be used. In order to conform to the baseline profile, this flag must equal 0, specifying that only CAVLC entropy encoding is allowed. If this flag equalled 1, CABAC entropy encoding would be indicated as allowed by the main profile.

pic_order_present_flag If this flag is equal to 1, it indicates that the picture order count syntax elements are present in the slice headers, and when the flag is 0 it states that the elements are not present in the slice headers.

num_slice_groups_minus1 This parameter specifies the number of slice groups minus 1 are used for a picture. If this parameter equals 0, this indicates that all of the slices of a picture belong to the same single slice group, meaning that FMO is not permitted for this picture. For the baseline profile, this parameter may be any value between 0 and 7 for map types 0, 1, 2 and 6. The following parameters may also be used where FMO is employed.

slice_group_map_type This parameter specifies how the slice group map units relate to the slice groups and may be any value between 0 and 6.

slice_group_map_type = 0 Uses Interleaved slice groups, whereby macroblocks are assigned in turn for each row length to the required number of slice groups.

run_length_minus1[i] This parameter is used for interleaving slice groups to indicate the number of consecutive slice groups to be assigned to the *ith slice group*, where *i* refers to the `num_slice_groups_minus1` variable.

slice_group_map_type = 1 Uses Dispersed slice groups. Each individual macroblock is assigned in turn to the next slice group.

slice_group_map_type = 2 Specifies one or more foreground groups and a left-over or background slice group. Varying sized rectangular regions form slice

groups, and those macroblocks not allocated to a specific foreground group, will automatically be allowed to the leftover group.

top_left[i]

bottom_right[i] These parameters are used to represent the top_left and bottom_right corners of the rectangular regions.

slice_group_map_type = 3 This type is known as box-out, and creates a box starting from the centre of the frame to a specific size forming one slice group, and the rest of the macroblock blocks form another slice group.

slice_group_change_direction_flag When this flag is 1, indicates that a counter-clockwise box-out is to be used.

change_rate_minus1 This parameter specifies the number of slice group map units that the size of a slice group can change from one picture to the next.

slice_group_map_type = 4 A Raster Scan is used to form this slice group, whereby a specific number of macroblocks starting from the top-left, moving left to right, top to bottom form a slice group, and the rest of the macroblocks form another slice group.

slice_group_change_direction_flag If this flag is equal to 1, in indicates to use a reverse raster scan.

change_rate_minus1 Again this parameter specifies the number of slice group map units that the size of a slice group can change from one picture to the next.

slice_group_map_type = 5 This type is known as a wipe, and is similar to a raster scan, but uses a vertical scan, starting again from the top left, but moving from top to bottom, left to right.

slice_group_change_direction_flag As with the raster scan, when this flag is equal to 1, a reverse direction is indicated, being to wipe left, as opposed to right.

change_rate_minus1 Again this parameter specifies the number of slice group map units that the size of a slice group can change from one picture to the next.

slice_group_map_type = 6 This type indicates that slice group map units are mapped to specific slice groups, and this map is user defined.

pic_size_in_map_units_minus1 This parameter is used to indicate the number of slice group map units that are in the picture.

slice_group_id This parameter is used to identify a slice group.

num_ref_idx_l0_active_minus1 This parameter specifies the maximum reference index for reference picture list 0 that should be used to decode each slice of the picture.

num_ref_idx_l1_active_minus1 This parameter is not required to conform to the baseline profile, as it is used specifically for B slices, however it is used similarly to above.

weighted_pred_flag This parameter is used to indicate whether weighted prediction should be applied to P and SP slices, however in order to conform to the baseline profile, this flag has been hardcoded to 0, meaning that it should not be applied to P slices.

weighted_bipred_idc This parameter when equal to 0 indicates that weighted prediction should not be applied to B slices, and again to conform to the baseline profile, this parameter must be 0.

pic_init_qp_minus26 This syntax element indicates the initial value minus 26 of the quantisation parameter for each slice. This value is hardcoded to 0, as the actual quantisation parameter used in the encoding exists in the slice header.

pic_init_qs_minus26 This syntax element specifies the initial value for SP or SI slices, and its value is therefore not required in order to conform to the baseline profile. As well as this, it has been hardcoded to 0 as the slice header contains the actual quantisation parameter used.

chroma_qp_index_offset This parameter provides the offset that shall be added to the quantisation parameters for colour alignment.

deblocking_filter_control_present_flag When 1, this flag is used to specify that the deblocking filter syntax elements are present in the slices header, but when this flag is 0, the inferred values are in effect.

constrained_intra_pred_flag When this flag equals 1, prediction of macroblocks using intra macroblock prediction modes only use residual data and decoded samples from I or SI macroblocks. When 0, the usage of residual data and decoded inter macroblock predicted samples may also be used.

redundant_pic_cnt_present_flag When this flag is equal to 1, it indicates that the `redundant_pic_cnt` syntax element is present in all slice headers and data partitions, when 0, it is only present in data partition A.

D.3 Bitstream info

A raw byte sequence payload (`rbsp`) is created to be filled by the PPS payload.

The first Syntax Element of the PPS NALU, is `pic_parameter_set_id`. This element is currently hardcoded to 0 and is parsed using `ue_v()` function. The `ue_v()` parsing process is discussed in Appendix C.

The next syntax element to be parsed is the `seq_paramater_set_id` is also hardcoded to 0 parsed using `ue_v()`.

The `entropy_coding_mode_flag` and the `pic_order_present_flag` are parsed using the `u_1` function. Both of these parameters have been hardcoded to 0.

The syntax element `num_slice_groups_minus1` is parsed using `ue_v`, and if this value is allowed to be 0, it means that all slices of the picture belong in the same slice group and that FMO is not employed.

The syntax elements of `num_ref_l0_active_minus1` is the number of reference frames requested by the user for inter motion search. This value is defaulted to 1 in the GUI therefore minus 1, a value of 0 is used for this parameter. The next syntax element, `num_ref_l0_active_minus1` is hardcoded to 0, as it is not required for the baseline profile. Both of these parameters are parsed using `ue(v)`, and therefore each return a value of 1.

The `weighted_pred_flag` is the next syntax element to be placed on the bitstream and has been hardcoded to 0, in order to conform to the baseline profile. It is parsed using the `u_1()` function.

The `weighted_bipred_flag` is also hardcoded to 0 to conform to the baseline profile however is parsed using `u(v)` as 2 bits are used to describe this syntax element.

The parsing process required by the `pic_init_qp_minus26` uses the `se(v)` descriptor, meaning that it is a signed integer value as its Exp-Golomb-coded syntax element. Being a signed integer value, means that both negative and positive values need to be taken into consideration. Initially the parameter value will be doubled if a negative value and doubled and reduced by 1 for a positive value. This value is Exp-Golomb coded as per the descriptor `ue(v)` for unsigned integer values. If this parameter is 0, this parsing process will add a 1 to the end of the current bitstream.

The next two parameters, being `pic_init_qs_minus26` and `chroma_qp_index_offset` both also have a signed integer value that needs to be Exp-Golomb coded. I have used a default value of 0 for both as the quantisation parameters exist in the slice header and these values will not be used.

The next three parameters are `deblocking_filter_control_present_flag`, `constrained_intra_pred_flag` and `redundant_pic_cnt_present_flag` and all use the `u_1()` parsing process. I have used a value of 0 as the default in the GUI for each of these parameters. All of the necessary PPS parameters have now been included in the bitstream.

As the sample PPS bitstream shown in the next section finished neatly at the completion of the byte, it is necessary to finish of the `rbsp` with a string of data bits, (`sodb`). In this case the `msb` of the byte is set to 1 and the remaining bits are set to 0. The `rbsp` is now converted to a `nal_u`.

The first byte of the `nal_u` buffer is filled with `nal_u` header. The 7th bit is 0, which is the hardcoded value of the `nal_u_forbidden_bit`. The 5th bit becomes the `nal_u_reference_idc` value of 3, and as this value is not 0, it infers that the NAL unit does not contain a slice or a slice data partition that is part of a non-reference picture. Finally the bit is used to represent the `nal_u_unit_type`, which is type 8, which indicates that it is of type

`pps_parameter_set_rbsp.`

Using Annex B byte stream format, a long start code length is specified to be used for PPS NALUs. This means that the start code prefix length will be 4 bytes long.

The start code is then written to the relevant output file, as specified in the `encoder.cfg`, followed by the PPS NALU header byte and the 3 bytes of the PPS.

In total using the GUI's default parameter values, there have been 64 bits written for the PPS NALU, including the start code, the NALU header and the PPS.

D.4 PPS NALU Bitstream

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

Long Start Code - 4 bytes

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1

PPS NALU Header

0	1	1	0	1	0	0	0
---	---	---	---	---	---	---	---

PPS Byte 1 = (206)

1	1	0	0	1	1	1	0
---	---	---	---	---	---	---	---

PPS Byte 2 $\bar{8}$ (56)

0	0	1	1	1	0	0	0
---	---	---	---	---	---	---	---

PPS Byte 3 = (128)

1	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

Appendix E

VCL Bitstream

E.1 Chapter Overview

The Network Access Layer Unit, NALU, that immediately follows the Sequence Parameter Set, SPS) and the Picture Parameter Set, PPS contains the first picture data for the video. The first NALU must be an Intra picture, which is a picture that does not reference any other pictures in the video sequence.

This NALU follows the encoding of the first frame from the `container_qcif.yuv` video sequence, sourced from the Kansas State University.

E.1.1 Slice Header Bitstream

The first syntax element of a slice header is the `current_mb_nr`, which for an IDR picture would be equal to 0. This syntax element is parsed using the `ue(v)` descriptor for unsigned integer Exp-Golomb coded syntax elements.

The next slice header parameter is the `slice_type`. This parameter is parsed using the `ue(v)` descriptor and as it is an I Slice, it has been given a value of 7.

The next parameter to be included is the reference to the appropriate picture parameter

set, PPS. The `pic_parameter_set_id` is hardcoded to 0 and is parsed using `ue(v)`.

The `frame_num`, represents the frame number and is the next parameter to be placed into the bitstream, and is parsed using the `u(n)` function.

The next parameter to be placed onto the slice header is the `idr_pic_id` which is hardcoded to 0 and parsed using `ue(v)`.

The slice header parameter `pic_order_cnt_lsb` is next to be placed onto the bitstream, has a value of 0 and is parsed using `u(n)`.

The next two slice header parameters are `no_output_of_prior_pics_flag` and `long_term_reference_flag`, and are used for reference pictures. Both of these parameters are 0 and parsed using the function `u_1()`.

The quantisation parameter is now inserted into the slice header, and is known as `slice_qp_delta`. This parameter is determined by the quantisation parameter provided by the user in the Graphical User Interface (GUI). The default values has been set to 16, minus 26 minus `pic_init_qp_minus26`. If `pic_init_qp_minus26` equals 0 the quantisation parameter is will be -10. This parameter is parsed using the `se(v)` descriptor for signed integers.

This completes the Slice Header with a length of 41 bits.

E.1.2 Macroblock Header Bitstream

Prior to the macroblock header being added to the bitstream, the prediction values and coefficients are all determined for the first block.

The first macroblock has a type of I4MB, which has a defined value for `mb_type` of 9, and this is changed to a value of 0 for coding. This parameter is parsed using `ue(v)` and therefore a 1 is added to the bitstream. This value is inserted as the next bit in the current byte, therefore immediate following slice header information.

The next syntax element to be parsed is of type `SE_INTRAPREDMODE`, and is the

first value of the intra prediction mode array, which has been hardcoded entirely to -1 as the macroblock is of type I4MB. The -1 is converted to 1 and is allocated 1 bit of the bitstream. The rest of 15 bits of the intra prediction mode array is parsed in a similar format, with the addition of 15 more 1's to the bitstream.

The next parameter to be added to the bitstream is `intra_chroma_pred_mode` and specifies the type of spatial prediction to be used for chroma. In this instance the value is 0, meaning DC prediction.

If this macroblock contained motion information it would be included into the bitstream at this point.

The next parameter to be added to the bitstream is the `coded_block_pattern`, CBP, which is of type `SE_CBP_INTRA` and has a value of 31. This CBP value, in conjunction with the whether the source is of YUV or RGB format determines the `cbp`.

The next parameter to be put onto the bitstream is `mb_qp_delta` which is the difference between the luma quantisation parameter of the current block and the previous macroblock's luma quantisation parameter. This has a value of 0 and is parsed using the `ue(v)`.

E.1.3 Macroblock Image Bitstream

The `nnz` value is now predicted for the luma component where the prediction from the Neighboring Blocks for the Number of Nonzero Coefficients for Luma Blocks is sought. Given that this is the first macroblock, the `nnz` value for both the luma and chroma components is 0.

The value of 7, which is taken from the vlc code table is incorporated into the bitstream, using 10 bits.

The specific reference point to the length and vlc code tables is determined by: if the `nnz` value is less than 2 table 0 is used, the number of trailing ones that are required, in this case 0, determines that row 0 is used and finally the column reference is 4 which

is the coefficient token when `chroma_dc` prediction is used. The length vlc table is also referenced to determine the number of bits to allocate to the vlc code. The variable length coding for this entropy coding method has been evaluated.

The coefficient levels now are VLC coded, with the number of bits used in the bitstream and the coefficient's value determined by the level. The coefficient levels have previously been developed from the original luma pixel value following horizontal and vertical transforms and then quantised. If this level is equal to -28, the number of bits to be used to represent the value is 28 bits as the absolute value of the level is greater than 17. If less than 17 but greater than 9, 19 bits will be used, and if 8 or less, the absolute level value multiplied by 2, minus 1 if the original level value was positive.

The coded coefficient value is determined by $4096 + (\text{absolute level value} - 17) * 2 + 1$. In this case the coefficient value is now 4119, and will be written to the bitstream using 28 bits.

The next coefficient level to be VLC coded is -37 and also requires 28 bits. The coefficient value to be written to the bitstream is 4109 and was determined using the same method as before.

The third coefficient level is -36, and after VLC coding becomes the value of 15, and requires 12 bits.

The coefficient level 78 is the next value that undergoes VLC coding, becoming 26, requiring 14 bits on the bitstream. The absolute value of the coefficient is determined to be 78, and the sign has a value of 0, indicating a positive value. As this is the 4th vlc variable, 15 is multiplied by 8, (given by $2^{\text{the vlc number} - 1}$ plus 1, equalling 121. One is then taken from the absolute coefficient value, and divided by 8, therefore giving 9. Given than 78 is less than 121, the number of bits required by the coefficient is determined by $9 + \text{the vlc number} + 1$ equalling 12. A suffix is then determined by binary anding the absolute coefficient value minus 1, with minus 1 from $2^{\text{vlc number}}$, which equals 5. The codeword of 26 is finally determined by the vlc number multiplied by 4 plus the sign value plus suffix value multiplied by 2.

The next parameter to be encoded is the total zeros syntax element, which has a value

of 6, and a vlc number of 3. These values are used to reference a CAVLC total zeros table for the total number of bits to be used and coded total vlc variable. In this instance the total zeros parameter equals 4, and will use a length of 3 bits.

The next parameter to be encoded is the run_before syntax element, which has a value of 5, and again the vlc number is 3. A CAVLC run table is referenced for both the length in bits and the code value. The encoded value now becomes 5, which is allowed a length of 3 bits.

The next run_before syntax element is now encoded using the table for the bitstream. Originally it had a value of 0, and a vlc number of 0. Following encoding, a single 1 is placed on the bitstream.

The third run_before parameter is now encoded from a 1 value and vlc number of 0 to a 0 value, with length of 1.

The next parameter to be used in the bitstream is the coeff_token, used to specify the total number of non-zero transform coefficient levels. A CAVLC table is referenced, and the value of 8 results to be put onto the bitstream, using 6 bits.

The next three 4×4 blocks of the macroblock are now written in the same format as before, whereby the CAVLC changed coefficient levels, the total number of zeros and the number of consecutive transform coefficient levels in the scan.

The chroma information is now determined and written to the bitstream. The syntax element is now of type SE_CHR_DC_INTRA. As with the luma component of the macroblock, the first parameter to be added to the bitstream is for the coeff_token. The VLC table used to generate this parameter's value and length is different to the one used to generate the luma coefficient token value.

The first chroma coefficient contains the level_prefix and level_suffix parameters, whereby a value of 4096 is derived and added to the bitstream, using 28 bits.

Following the coefficient is the total_zeros parameter for U chroma. If the total_zeros parameter is a 1, a 1 is output to the bitstream.

The V chroma information is now prepared for the bitstream. The `coeff_token` is determined using the chroma `coeff_token` table, and the value of 6, with a length of 6 bits is derived from the reference information.

The `trailing_ones_sign_flag` is given a 1 value to be placed onto the bitstream. This flag indicates that the corresponding transform coefficient level should be decoded as -1.

The V chroma coefficient was previously determined to have a value of -4 for the first macroblock of the sample frame and after encoding requires the value of 1 to be placed onto the bitstream over 6 bits.

The `total_zeros` parameter for the V chroma component is now encoded for the bitstream, and is a 1 which is to be represented over 2 bits. Finally, the `run_before` parameter is added to the bitstream for the V chroma component, whereby the value of 0 will be represented by 1 bit.

The first macroblock has now been written to the bitstream. The bit counters and macroblock addressing and counters are all reset and updated, in order to prepare for the next macroblock. In total 510 bits were written to encode this macroblock.

The next macroblock is now encoded for the bitstream, until the entire slice is written. Given that the number of slice groups to be used is set to 0, flexible macroblock ordering, FMO, is not employed, therefore macroblocks are written in a raster scan order, from left to right, top to bottom.

The second macroblock will now have a neighbour, being the neighbour to its left, and this neighbour can be used in the determination of any applicable motion vectors. A prediction value is determined for the U chroma block, being the average value of the U chroma coefficients from the previous block. The U chroma average for the previous macroblock was 137, therefore this is the predicted value for the next chroma coefficient. This is known as horizontal prediction, and is termed `HOR_PRED_8`. If the above macroblock was available, vertical prediction may also occur, and when both horizontal and vertical macroblocks are available, plane prediction will also be used.

The cost between the original value and the predicted value is now determined, given

that the original value is 136 for the U chroma component, and the predicted value of 137 exists, therefore there is a difference of -1 between these values. Summing up all of these differences that were created using DC_PRED_8 for the 4x4 block is 12.

The difference between the original value and the predicted value for the V chroma component, using DC_PRED_8 is also determined to be -1 for the first pixel, whereby the original value is 125, and the predicted value is 126 and for the entire 4x4 block, a difference of 14 exists. The entire macroblock cost for this type of prediction for both U and V chroma values is 97.

The cost is now determined using horizontal prediction. For the first U chroma component a difference of -1 again exists between the original value and predicted value and again a cost of 12 is associated with the absolute value of all of the differences for this macroblock. The same values exist for the V chroma component with a difference of -1 between the original and horizontal predicted values. For the 4x4 V chroma block, a difference of 14 exists and 97 is the cost for the U and V chroma values for this macroblock.

As the cost associated with the horizontal prediction method is the same as the direct prediction method, the DC_PRED_8 method will be used as the chroma intra prediction mode.

The Y values from the previous macroblock are averaged to determine the value to be used for the DC luma prediction, giving $(106 + 255 + 231 + 232)/4 = 206$. Horizontal luma prediction is now effected, whereby values are predicted using various algorithms on the values of the previous macroblock.

Using DC_PRED mode the difference for the first pixel of this macroblock is -100, and for the entire 4x4 macroblock it is 791.

The luma component is now transformed first horizontally and then vertically and subsequently quantised. The coefficient for the first luma pixel of this macroblock is -64. This is continued for the rest of the macroblock.

The macroblock header information is now written to the bitstream for the second

macroblock, beginning with the macroblock type, which is I4MB, therefore a 1 is written to the bitstream.

As with the first macroblock, the header is continued to be written with the `intra_pred_mode` syntax elements followed by the `intra_chroma_pred_mode` syntax element, continuing through to write the necessary coefficients and finally the `run_before_VLC` parameter. This second macroblock finishes with 126 bytes being written to the bitstream.

In total there are 99 macroblocks written in the first I Slice. A string of 0 data bits is used to complete the final byte in the slice and if the slice was completed on a full slice, a complete zero byte would be inserted. The Slice is now in an RBSP format. The final byte of this slice is at byte 9788. This includes the slice and macroblock headers, as well as the actual data.

The RBSP stream is now checked in order to ensure that accidental start code emulation does not occur within the NALU payload.

Deblocking is then conducted on each macroblock in order within the frame on the vertical and then horizontal edges.

The distortion is determined for each of the components, with 38269 being determined for the y component, 5688 for the u component and 5782 for the v component.

A NALU is created for the Intra Slice, of size 101376. A `rbsp` is created to be filled by the PPS payload, and it is this payload that will be used in the NALU as the data information.

The first byte of the `nal_u` buffer is filled with `nal_u` header. The 7th bit is 0, which is the hardcoded value of the `nal_u_forbidden_bit`. The 5th bit becomes the `nal_u_reference_idc` value of 3, and as this value is not 0, it infers that the NAL unit does not contain a slice or a slice data partition that is part of a non-reference picture. Finally four bits are used to represent the `nal_u_unit_type`, which is type 5 indicating type `slice_layer_without_partitioning_rbsp`.

As I am using an Annex B byte stream format, a long start code length is specified to be used for IDR NALUs. This means that the start code prefix length will be 4 bytes long.

The start code is then written to the relevant output file, as specified in the encoder.cfg, followed by the, NALU header byte and the 9788 bytes of payload.

There have therefore been 9792 bytes written for the first IDR NALU, including the start code, the NALU header, Slice and Macroblock headers, and 99 Macroblocks of image information.

E.2 Intra VCL NALU Bitstream

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

Long Start Code - 4 bytes

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1

Slice 1 NALU Header

0	1	1	0	0	1	0	1
---	---	---	---	---	---	---	---

Intra Slice Header Byte 1 = (136)

1	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

Intra Slice Header Byte 2 = (128)

1	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

Intra Slice Header Byte 3 = (32)

0	0	1	0	0	0	0	0
---	---	---	---	---	---	---	---

Intra Slice Header Byte 4 = (0)

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

Intra Slice Header Byte 5 = (10)

0	0	0	0	1	0	1	0
---	---	---	---	---	---	---	---

Intra Slice Header Byte 6 (MSB only) and Macroblock Header Byte 1 = (255)

1	1	1	1	1	1	1	1
---	---	---	---	---	---	---	---

Macroblock Header Byte 2 = (255)

1	1	1	1	1	1	1	1
---	---	---	---	---	---	---	---

Macroblock Header Byte 3 = (234)

1	1	1	0	1	0	1	0
---	---	---	---	---	---	---	---

Macroblock Image Byte 4 = (3)

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

Macroblock Image Byte 5 (MSB only) and First Coefficient value for bitstream = (128)

1	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

Macroblock Image Byte 6,7 and First Coefficient values continued= (0), - (128)

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Macroblock Image Byte 8 and First Coefficient (5 bits), Second Coefficient (3 bits) = (184)

1	0	1	1	1	0	0	0
---	---	---	---	---	---	---	---

Macroblock Image Byte 9, 10, 11 and Second Coefficient (24 bits) continued = (0) - (8) - (6)

0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	1	1	0

Macroblock Image Byte 12, and Third Coefficient (7 bits) = - (128)

1	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

Macroblock Image Byte 13, 14, Rest of Third Coefficient(5 bits) and Fourth Coefficient (11 bits)= x(120) - (3)

0	1	1	1	1	0	0	0
0	0	0	0	0	1	1	1

Macroblock Image Byte 15, Fourth Coefficient (3 bits), total_zeros (3 bits) run_before (2 bits) = r (82)

0	1	0	1	0	0	1	0
---	---	---	---	---	---	---	---

Macroblock Image Byte 16, run_before = - (196)

1	1	0	0	0	1	0	0
---	---	---	---	---	---	---	---

Macroblock Image Byte 17 = Completed by subsequent Luma Information for Macroblock 1 - ()

The next 3 8x8 luma blocks continue for another 45 bytes after this one.

0

Macroblock Image Byte 63, U Chroma Coefficient token(4 lsb) = - (177)

1	0	1	1	0	0	0	1
---	---	---	---	---	---	---	---

Macroblock Image Byte 64, = - (192)

1	1	0	0	0	0	0	0
---	---	---	---	---	---	---	---

Macroblock Image Byte 65 = - (64)

0	1	0	0	0	0	0	0
---	---	---	---	---	---	---	---

Macroblock Image Byte 66 = 4 (52)

0	0	1	1	0	1	0	0
---	---	---	---	---	---	---	---

Macroblock Image Byte 67 = - (21)

0	0	0	1	0	1	0	1
---	---	---	---	---	---	---	---