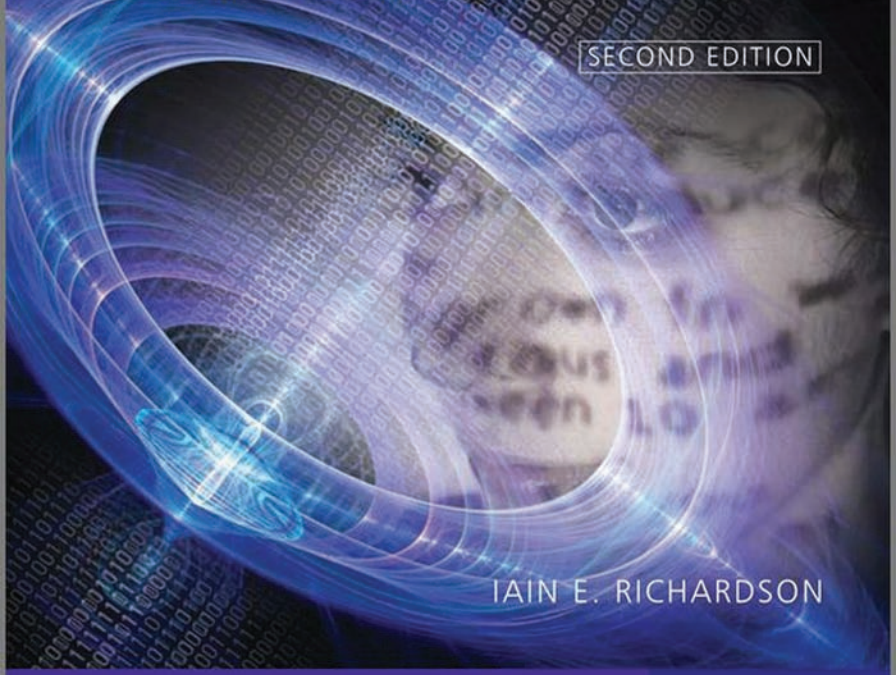
THE H.264 ADVANCED VIDEO COMPRESSION STANDARD

SECOND EDITION



IAIN E. RICHARDSON

WILEY

AMAZON-1009
Amazon.com v. InterDigital
IPR2026-00192

THE H.264 ADVANCED VIDEO COMPRESSION STANDARD

THE H.264 ADVANCED VIDEO COMPRESSION STANDARD

Second Edition

Iain E. Richardson
Vcodex Limited, UK

WILEY
A John Wiley and Sons, Ltd., Publication

This edition first published 2010
© 2010, John Wiley & Sons, Ltd

First Edition published in 2003

Registered office
John Wiley & Sons Ltd, The Atrium, Southern Gate, Chichester, West Sussex, PO19 8SQ, United Kingdom

For details of our global editorial offices, for customer services and for information about how to apply for permission to reuse the copyright material in this book please see our website at www.wiley.com.

The right of the author to be identified as the author of this work has been asserted in accordance with the Copyright, Designs and Patents Act 1988.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, except as permitted by the UK Copyright, Designs and Patents Act 1988, without the prior permission of the publisher.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic books.

Designations used by companies to distinguish their products are often claimed as trademarks. All brand names and product names used in this book are trade names, service marks, trademarks or registered trademarks of their respective owners. The publisher is not associated with any product or vendor mentioned in this book. This publication is designed to provide accurate and authoritative information in regard to the subject matter covered. It is sold on the understanding that the publisher is not engaged in rendering professional services. If professional advice or other expert assistance is required, the services of a competent professional should be sought.

Library of Congress Cataloguing-in-Publication Data

Richardson, Iain E. G.
The H.264 advanced video compression standard / Iain E. Richardson. – 2nd ed.
p. cm.
Rev. ed. of: H.264 and MPEG-4 video compression. c2003.
Includes bibliographical references and index.
ISBN 978-0-470-51692-8 (pbk.)
1. Digital video–Standards. 2. Video compression–Standards. 3. MPEG (Video coding standard) 4. Multimedia systems. I. Richardson, Iain E. G. H.264 and MPEG-4 video compression. II. Title.
TK6680.5.R52 2010
006.6′96–dc22
2009054387

A catalogue record for this book is available from the British Library.

ISBN: 978-0-470-51692-8

Typeset in 10/12pt Times by Aptara Inc., New Delhi, India
Printed and bound in Great Britain by CPI Autony Rowe, Chippenham, Wiltshire

To Pat

Language is living,
but what is most important
goes deeper than words.

Contents

About the Author **xiii**

Preface **xv**

Glossary **xvii**

List of Figures **xxi**

List of Tables **xxix**

1 Introduction **1**
1.1 A change of scene 1
1.2 Driving the change 4
1.3 The role of standards 4
1.4 Why H.264 Advanced Video Coding is important 4
1.5 About this book 5
1.6 Reference 6

2 Video formats and quality **7**
2.1 Introduction 7
2.2 Natural video scenes 7
2.3 Capture 8
2.3.1 Spatial sampling 9
2.3.2 Temporal sampling 9
2.3.3 Frames and fields 11
2.4 Colour spaces 12
2.4.1 RGB 12
2.4.2 YCrCb 13
2.4.3 YCrCb sampling formats 14
2.5 Video formats 16
2.5.1 Intermediate formats 16
2.5.2 Standard Definition 17
2.5.3 High Definition 18
2.6 Quality 19
2.6.1 Subjective quality measurement 20
2.6.2 Objective quality measurement 21

2.7 Summary 24
2.8 References 24

3 Video coding concepts 25
3.1 Introduction 25
3.2 Video CODEC 26
3.3 Prediction model 28
3.3.1 Temporal prediction 28
3.3.2 Spatial model: intra prediction 38
3.4 Image model 40
3.4.1 Predictive image coding 41
3.4.2 Transform coding 42
3.4.3 Quantization 50
3.4.4 Reordering and zero encoding 52
3.5 Entropy coder 57
3.5.1 Predictive coding 57
3.5.2 Variable-length coding 58
3.5.3 Arithmetic coding 65
3.6 The hybrid DPCM/DCT video CODEC model 68
3.7 Summary 79
3.8 References 79

4 What is H.264? 81
4.1 Introduction 81
4.2 What is H.264? 81
4.2.1 A video compression format 81
4.2.2 An industry standard 82
4.2.3 A toolkit for video compression 83
4.2.4 Better video compression 83
4.3 How does an H.264 codec work? 83
4.3.1 Encoder processes 85
4.3.2 Decoder processes 89
4.4 The H.264/AVC Standard 91
4.5 H.264 Profiles and Levels 92
4.6 The H.264 Syntax 94
4.7 H.264 in practice 97
4.7.1 Performance 97
4.7.2 Applications 98
4.8 Summary 98
4.9 References 98

5 H.264 syntax 99
5.1 Introduction 99
5.1.1 A note about syntax examples 99
5.2 H.264 syntax 100
5.3 Frames, fields and pictures 101

5.3.1	Decoding order	104
5.3.2	Display order	104
5.3.3	Reference picture lists	106
5.3.4	Frame and field coding	111
5.4	NAL unit	114
5.5	Parameter Sets	115
5.6	Slice layer	117
5.6.1	Slice types	117
5.6.2	Slice header	117
5.6.3	Slice data	118
5.7	Macroblock layer	119
5.7.1	Overview	119
5.7.2	The Intra PCM mode	121
5.7.3	Macroblock prediction	122
5.7.4	Residual data	124
5.7.5	Macroblock syntax examples	127
5.8	Summary	134
5.9	References	135
6	**H.264 Prediction**	**137**
6.1	Introduction	137
6.2	Macroblock prediction	137
6.3	Intra prediction	138
6.3.1	4×4 luma prediction modes	143
6.3.2	16×16 luma prediction modes	146
6.3.3	Chroma prediction modes	147
6.3.4	8×8 luma prediction, High profiles	148
6.3.5	Signalling intra prediction modes	148
6.4	Inter prediction	149
6.4.1	Reference pictures	151
6.4.2	Interpolating reference pictures	152
6.4.3	Macroblock partitions	157
6.4.4	Motion vector prediction	158
6.4.5	Motion compensated prediction	162
6.4.6	Inter prediction examples	164
6.4.7	Prediction structures	169
6.5	Loop filter	171
6.5.1	Boundary strength	172
6.5.2	Filter decision	173
6.5.3	Filter implementation	174
6.5.4	Loop filter example	174
6.6	Summary	177
6.7	References	177
7	**H.264 transform and coding**	**179**
7.1	Introduction	179

7.2 Transform and quantization 179
7.2.1 The H.264 transforms 179
7.2.2 Transform processes 180
7.2.3 Integer transform and quantization: 4 × 4 blocks 185
7.2.4 Integer transform and quantization: 8 × 8 blocks 198
7.2.5 DC transforms 203
7.2.6 Transform and quantization extensions in the High profiles 204
7.3 Block scan orders 206
7.4 Coding 207
7.4.1 Exp-Golomb Coding 208
7.4.2 Context Adaptive Variable Length Coding, CAVLC 210
7.4.3 Context Adaptive Binary Arithmetic Coding, CABAC 217
7.5 Summary 220
7.6 References 221

8 H.264 conformance, transport and licensing 223
8.1 Introduction 223
8.2 Conforming to the Standard 223
8.2.1 Profiles 224
8.2.2 Levels 226
8.2.3 Hypothetical Reference Decoder 230
8.2.4 Conformance testing 236
8.3 H.264 coding tools for transport support 237
8.3.1 Redundant slices 237
8.3.2 Arbitrary Slice Order (ASO) 238
8.3.3 Slice Groups/Flexible Macroblock Order (FMO) 238
8.3.4 SP and SI slices 240
8.3.5 Data partitioned slices 243
8.4 Transport of H.264 data 244
8.4.1 Encapsulation in RBSPs, NALUs and packets 244
8.4.2 Transport protocols 245
8.4.3 File formats 247
8.4.4 Coding and transport issues 247
8.5 Supplemental Information 248
8.5.1 Supplemental Enhancement Information (SEI) 248
8.5.2 Video Usability Information (VUI) 248
8.6 Licensing H.264/AVC 248
8.6.1 Video coding patents 250
8.6.2 Video coding standards and patents 252
8.6.3 Licensing H.264/AVC patents 252
8.7 Summary 253
8.8 References 253

9 H.264 performance 255
9.1 Introduction 255
9.2 Experimenting with H.264 256

	9.2.1 The JM Reference Software	256
	9.2.2 Other software encoders/decoders	261
	9.2.3 H.264 stream analysis	263
9.3	Performance comparisons	265
	9.3.1 Performance criteria	265
	9.3.2 Performance examples: Foreman sequence, QCIF resolution	265
	9.3.3 Performance examples: Foreman and Container sequences	269
	9.3.4 Performance examples: Inter prediction structures	271
	9.3.5 Performance example: H.264 vs. MPEG-4 Visual	273
9.4	Rate control	274
	9.4.1 Rate control in the JM reference encoder	276
9.5	Mode selection	279
	9.5.1 Rate Distortion Optimized mode selection	281
9.6	Low complexity coding	283
	9.6.1 Approximating the cost function	283
	9.6.2 Reducing the set of tested modes	284
	9.6.3 Early termination	285
9.7	Summary	285
9.8	References	285
10	**Extensions and directions**	**287**
10.1	Introduction	287
10.2	Scalable Video Coding	288
	10.2.1 Simulcast transmission	288
	10.2.2 Scalable transmission	289
	10.2.3 Applications of Scalable Video Coding	290
	10.2.4 Scalable Video Coding in H.264	290
	10.2.5 Temporal scalability	292
	10.2.6 Quality scalability: overview	294
	10.2.7 Spatial scalability: overview	294
	10.2.8 Spatial scalability in detail	294
	10.2.9 Quality scalability in detail	298
	10.2.10 Combined scalability	299
	10.2.11 SVC performance	299
10.3	Multiview Video Coding	302
	10.3.1 H.264 Multiview Video Coding	304
10.4	Configurable Video Coding	306
	10.4.1 MPEG Reconfigurable Video Coding	307
	10.4.2 Fully Configurable Video Coding	308
10.5	Beyond H.264/AVC	310
10.6	Summary	310
10.7	References	311
Index		**313**

About the Author

Professor Iain Richardson is an internationally known expert on the MPEG and H.264 video compression standards.

The author of *H.264 and MPEG-4 Video Compression*, a widely cited work in the research literature, Professor Richardson has written two further books and over 70 journal and conference papers on video compression. He regularly advises companies on video compression technology, video coding patents and company acquisitions in the video coding industry. Professor Richardson leads an internationally renowned video coding research team, contributes to the MPEG industry standards group and is sought after as an expert witness. Based in Aberdeen, Scotland, he regularly travels to the US and Europe.

Preface

The last decade has seen a quiet revolution in digital video technology. Digital video is everywhere: on our televisions, our DVD and Blu-Ray players, our computers, our music players and our mobile handsets. Only recently, a video image in a web page was an unusual sight. Nowadays, many of us are just as likely to catch the latest news on the web as on the TV. With the explosion of digital video applications, a billion-dollar industry has developed and expanded, with new companies and niche markets emerging, thriving and disappearing faster than anyone can easily track. Video compression is essential to all of these applications and markets, and the H.264 format is considered by many to be the state of the art in video compression.

When I wrote the first edition of this book in 2003, H.264 Advanced Video Compression had just been published as an International Standard and it was hard to predict its impact on industry. Its predecessor, MPEG-4 Visual, had arguably failed to live up to its promise, with only limited adoption in the market. Since 2003, the significant performance improvements that are built into H.264 have made it the clear successor to the older MPEG video standards in many applications, from mobile video to High Definition broadcasting. At the time of writing, the MPEG and VCEG standards committees are debating the possible successor to H.264. It is likely to be several years before a new standard is released, and several years after that before H.264 begins to become obsolete.

This book is intended to be a practical, accessible and unbiased guide to the H.264 video compression standard. As always, I have chosen to explain the details of H.264 in my own way, concentrating on what I feel is important to the engineer, researcher or student who needs a 'way in' to this complex yet important technical subject. This book is not the final word on H.264. By definition, that final word is provided by the standard itself and I advise any serious developer or implementer of H.264 to get hold of a copy of the standard. There is a need for a guidebook to the standard that explains the concepts, tools, benefits and disadvantages of the format, just as a good guidebook helps the tourist to get to know a foreign country and to become more at home there. Some visitors may be disappointed that their favourite subject is not covered in as much depth as they would like. I have made a deliberate choice to cover certain topics such as Scalable and Multiview Video Coding only briefly as they are still, in my view, in the early stages of practical implementation.

My sincere thanks to the many people who have helped to shape this book, including the readers of my earlier books who told me what they liked and what they wanted; the many companies and individuals who have asked me to solve their video compression problems; Kourosh Soroushian for discussions on Hypothetical Reference Decoders; Abharana Bhat,

Maja Bystrom, Sam Jansen, Sampath Kannangara and Yafan Zhao for reading and commenting on draft chapters; Gary Sullivan for many comments, corrections, suggestions and discussions; Nicky, Simone and the editorial team at John Wiley & Sons; and to Pat for reading the manuscript, cracking the whip and making me finish it.

I hope that you find the book useful; more importantly, I hope you enjoy it. Visit my website at www.vcodex.com and *tell me what you think*.

Iain Richardson
Aberdeen, 2010

Glossary

4:2:0 (sampling)	Sampling method: chrominance components have half the horizontal and vertical resolution of luminance component
4:2:2 (sampling)	Sampling method: chrominance components have half the horizontal resolution of luminance component
4:4:4 (sampling)	Sampling method: chrominance components have same resolution as luminance component
access unit	Complete coded frame or field
arithmetic coding	Coding method to reduce redundancy
artefact	Visual distortion in an image
ASO	Arbitrary Slice Order, in which slices may be coded out of raster sequence
block	Region of macroblock
block matching	Motion estimation carried out on rectangular picture areas
blocking	Square or rectangular distortion areas in an image
B slice	Coded slice predicted using bidirectional motion compensation
CABAC	Context-based Adaptive Binary Arithmetic Coding
CAVLC	Context Adaptive Variable Length Coding
chrominance or chroma	Colour difference component
CIF	Common Intermediate Format, a colour image format
CODEC	COder / DECoder pair
Coded Picture Buffer (CPB)	Buffer containing coded frames or fields
colour space	Method of representing colour images
DCT	Discrete Cosine Transform, a mathematical transform and/or its practical approximation(s)
direct prediction	A coding mode in which no motion vector is transmitted
DPCM	Differential Pulse Code Modulation

DSCQS	Double Stimulus Continuous Quality Scale, a scale and method for subjective quality measurement
DWT	Discrete Wavelet Transform
entropy coding	Coding method to reduce redundancy
error concealment	Post-processing of a decoded image to remove or reduce visible error effects
Exp-Golomb or ExpG	Exponential Golomb variable length codes
field	Odd- or even-numbered lines from an interlaced video sequence
FMO	Flexible Macroblock Order, in which macroblocks may be coded out of raster sequence
Full Search	A motion estimation algorithm
Fully Configurable Video Coding	A framework for video coding in which a codec may be completely re-configured during a communication session
GOP	Group of Pictures, a set of coded video images
H.261	A video coding standard
H.263	A video coding standard
H.264	A video coding standard
HDTV	High Definition Television
Huffman coding	Coding method to reduce redundancy
HVS	Human Visual System, the system by which humans perceive and interpret visual images
hybrid (CODEC)	CODEC model featuring motion compensation and transform
Hypothetical Reference Decoder (HRD)	Decoder 'model' that may be used to test bitstream conformance
IEC	International Electrotechnical Commission, a standards body
inter (coding)	Coding of video frames using temporal prediction or compensation
interlaced (video)	Video data represented as a series of fields
intra (coding)	Coding of video frames without temporal prediction
I slice	Slice coded without reference to any other frame
ISO	International Standards Organisation, a standards body
ITU	International Telecommunication Union, a standards body
JPEG	Joint Photographic Experts Group, a committee of ISO (also an image coding standard)
latency	Delay through a communication system
Level	A set of conformance parameters (applied to a Profile)

loop filter	Spatial filter placed within encoding or decoding feedback loop
luminance or luma	Monochrome or brightness component
Macroblock	Region of frame coded as a unit (usually 16×16 pixels in the original frame)
Macroblock partition	Region of macroblock with its own motion vector
Macroblock sub-partition	Region of macroblock with its own motion vector
motion compensation	Prediction of a video frame with modelling of motion
motion estimation	Estimation of relative motion between two or more video frames
motion vector	Vector indicating a displaced block or region to be used for motion compensation
MPEG	Motion Picture Experts Group, a committee of ISO/IEC
MPEG-1	A multimedia coding standard
MPEG-2	A multimedia coding standard
MPEG-4	A multimedia coding standard
MVC	Multiview Video Coding, in which multiple views of a scene may be jointly coded
NAL	Network Abstraction Layer
objective quality	Visual quality measured by algorithm(s)
Picture (coded)	Coded (compressed) video frame
P-picture (slice)	Coded picture (or slice) using motion-compensated prediction from one reference frame
profile	A set of functional capabilities (of a video CODEC)
progressive (video)	Video data represented as a series of complete frames
PSNR	Peak Signal to Noise Ratio, an objective quality measure
QCIF	Quarter Common Intermediate Format
quantize	Reduce the precision of a scalar or vector quantity
rate control	Control of bit rate of encoded video signal
rate-distortion	Measure of CODEC performance (distortion at a range of coded bit rates)
RBSP	Raw Byte Sequence Payload
RVC	Reconfigurable Video Coding, a framework for video coding in which a decoder may be constructed from pre-defined Functional Units.
RGB	Red/Green/Blue colour space
ringing (artefacts)	‘Ripple’-like artefacts around sharp edges in a decoded image
RTP	Real Time Protocol, a transport protocol for real-time data

scalable coding	Coding a signal into a number of layers
SVC	Scalable Video Coding
SI slice	Intra-coded slice used for switching between coded bitstreams (H.264)
slice	A region of a coded picture
SP slice	Inter-coded slice used for switching between coded bitstreams
statistical redundancy	Redundancy due to the statistical distribution of data
studio quality	Lossless or near-lossless video quality
subjective quality	Visual quality as perceived by human observer(s)
subjective redundancy	Redundancy due to components of the data that are subjectively insignificant
sub-pixel (motion compensation)	Motion-compensated prediction from a reference area that may be formed by interpolating between integer-valued pixel positions
test model	A software model and document that describe a reference implementation of a video coding standard
texture	Image or residual data
tree-structured motion compensation	Motion compensation featuring a flexible hierarchy of partition sizes
VCEG	Video Coding Experts Group, a committee of ITU
VCL	Video Coding Layer
video packet	Coded unit suitable for packetization
VLC	Variable Length Code
VLD	Variable Length Decoder
VLE	Variable Length Encoder
VLSI	Very Large Scale Integrated circuit
VQEG	Video Quality Experts Group
weighted prediction	Motion compensation in which the prediction samples from two references are scaled
YCrCb	Luminance/Red chrominance/Blue chrominance colour space

List of Figures

1.1	Video coding scenarios, one-way	3
1.2	Video coding scenario, two-way	3
2.1	Still image from natural video scene	8
2.2	Spatial and temporal sampling of a video sequence	8
2.3	Image with two sampling grids	9
2.4	Image sampled at coarse resolution (black sampling grid)	10
2.5	Image sampled at finer resolution (grey sampling grid)	10
2.6	Interlaced video sequence	11
2.7	Top field	12
2.8	Bottom field	13
2.9	Red, Green and Blue components of colour image	13
2.10	Cr, Cg and Cb components	14
2.11	4:2:0, 4:2:2 and 4:4:4 sampling patterns (progressive)	15
2.12	Allocation of 4:2:0 samples to top and bottom fields	17
2.13	Video frame sampled at range of resolutions	18
2.14	SD and HD formats	19
2.15	DSCQS testing system	21
2.16	PSNR examples: (a) Original; (b) 30.6dB; (c) 28.3dB	22
2.17	Image with blurred background (PSNR = 27.7dB)	22
3.1	Encoder/Decoder	26
3.2	Spatial and temporal correlation in a video sequence	26
3.3	Video encoder block diagram	27
3.4	Frame 1	29
3.5	Frame 2	29
3.6	Difference	29
3.7	Optical flow	30
3.8	Macroblock (4:2:0)	31
3.9	Motion estimation	32
3.10	Frame 1	33
3.11	Frame 2	33
3.12	Residual : no motion compensation	33
3.13	Residual : 16×16 block size	34
3.14	Residual : 8×8 block size	34
3.15	Residual : 4×4 block size	34

3.16 Close-up of reference region 35
3.17 Reference region interpolated to half-pixel positions 36
3.18 Integer, half-pixel and quarter-pixel motion estimation 36
3.19 Residual : 4 × 4 blocks, 1/2-pixel compensation 37
3.20 Residual : 4 × 4 blocks, 1/4-pixel compensation 37
3.21 Motion vector map : 16 × 16 blocks, integer vectors 38
3.22 Motion vector map : 4 × 4 blocks, 1/4-pixel vectors 39
3.23 Intra prediction: available samples 39
3.24 Intra prediction: spatial extrapolation 39
3.25 2D autocorrelation function of image 40
3.26 2D autocorrelation function of residual 41
3.27 Spatial prediction (DPCM) 41
3.28 4 × 4 DCT basis patterns 45
3.29 8 × 8 DCT basis patterns 45
3.30 Image section showing 4 × 4 block 46
3.31 Close-up of 4 × 4 block; DCT coefficients 47
3.32 Block reconstructed from (a) 1, (b) 2, (c) 3, (d) 5 coefficients 47
3.33 Two-dimensional wavelet decomposition process 48
3.34 Image after one level of decomposition 49
3.35 Two-stage wavelet decomposition of image 49
3.36 Five-stage wavelet decomposition of image 50
3.37 Scalar quantizers: linear; non-linear with dead zone 52
3.38 Vector quantization 53
3.39 8 × 8 DCT coefficient distribution (frame) 53
3.40 Residual field picture 54
3.41 8 × 8 DCT coefficient distribution (field) 54
3.42 Zigzag scan example : frame block 55
3.43 Zigzag scan example : field block 55
3.44 Wavelet coefficient and ‘children’ 57
3.45 Motion vector prediction candidates 58
3.46 Generating the Huffman code tree: Sequence 1 motion vectors 60
3.47 Huffman tree for sequence 2 motion vectors 61
3.48 MPEG4 TCOEF VLCs (partial) 64
3.49 Sub-range example 66
3.50 Arithmetic coding example 67
3.51 DPCM/DCT video encoder 69
3.52 DPCM/DCT video decoder 69
3.53 Input frame F_n 71
3.54 Reconstructed reference frame F'_{n-1} 71
3.55 Residual F_n-F'_{n-1} : no motion compensation 72
3.56 16 × 16 motion vectors superimposed on frame 73
3.57 Motion compensated reference frame 73
3.58 Motion compensated residual frame 74
3.59 Original macroblock : luminance 74
3.60 Residual macroblock : luminance 75
3.61 DCT coefficient magnitudes : top-right 8 × 8 block 76

3.62 Comparison of original and decoded residual blocks 78
3.63 Decoded frame F'_n 79
4.1 The H.264 video coding and decoding process 82
4.2 Video coding: source frames, encoded bitstream, decoded frames 83
4.3 Video codec: high level view 84
4.4 Typical H.264 encoder 84
4.5 Typical H.264 decoder 85
4.6 Prediction: flow diagram 86
4.7 Intra prediction 86
4.8 Original macroblock, intra prediction and residual 86
4.9 Inter prediction 87
4.10 Original macroblock, inter prediction and residual 87
4.11 Forward transform 88
4.12 Quantization example 88
4.13 Example: Block, transform coefficients, quantized coefficients 88
4.14 Rescaling example 89
4.15 Inverse transform: combining weighted basis patterns to create a 4×4 image block 90
4.16 Rescaled coefficients and inverse transform output 90
4.17 Reconstruction flow diagram 91
4.18 Profiles and Levels: example 93
4.19 H.264 syntax : overview 94
4.20 Syntax example: P-macroblock 95
4.21 P-macroblock decoding process 96
4.22 Residual luma and chroma coefficient blocks 97
4.23 A video frame compressed at the same bitrate using MPEG-2 (left), MPEG-4 Visual (centre) and H.264 compression (right) 97
5.1 Syntax overview 100
5.2 Picture handling in H.264, overview 103
5.3 Decoded Picture Buffer and picture orders 104
5.4 Display order: Type 0 example 105
5.5 Display order: Type 1 example 106
5.6 List 0 and List 1 ordering: example 108
5.7 Default reference pictures: example 108
5.8 Reference picture re-ordering syntax, simplified overview 109
5.9 Picture Adaptive Frame Field Coding example 112
5.10 Frame with macroblock pairs 113
5.11 MB pair coded using Macroblock Adaptive Frame Field Coding 113
5.12 Example: Sequence and Picture Parameter Sets 116
5.13 Macroblock layer syntax overview 119
5.14 mb_pred and sub_mb_pred syntax overview 123
5.15 Residual data syntax overview 125
5.16 Block scanning order, 4×4 transform, 4:2:0 sampling 125
5.17 Block scanning order, 8×8 luma transform, 4:2:0 sampling 126
5.18 Residual CAVLC block syntax overview 126
5.19 Residual CABAC block syntax overview 127

5.20	P macroblock, example 2	129
5.21	P macroblock example 4	131
5.22	B macroblock, example 5	133
6.1	Example of macroblock types and prediction sources	138
6.2	Intra prediction: adjacent blocks example	139
6.3	Intra prediction source samples, 4 × 4 or 8 × 8 luma blocks	140
6.4	Intra prediction source samples, chroma or 16 × 16 luma blocks	140
6.5	Example of intra block size choices, CIF, Baseline Profile. Reproduced by permission of Elecard.	141
6.6	QCIF frame with highlighted macroblock	142
6.7	Predicted luma frame formed using H.264 intra prediction	142
6.8	Residual after subtracting intra prediction	143
6.9	4 × 4 luma block to be predicted	144
6.10	Labelling of prediction samples, 4 × 4 prediction	144
6.11	4 × 4 intra prediction modes	144
6.12	Prediction blocks, 4 × 4 modes 0–8	145
6.13	Intra 16 × 16 prediction modes	146
6.14	16 × 16 macroblock	146
6.15	Prediction blocks, intra 16 × 16 modes 0–3	147
6.16	Intra mode prediction example	148
6.17	P macroblock prediction example	150
6.18	Example of integer and sub-pixel prediction	152
6.19	Current region	153
6.20	4 × 4 block to be predicted	153
6.21	Reference region	153
6.22	Prediction from integer samples	154
6.23	Reference region, half-pixel interpolated	154
6.24	Prediction from interpolated samples	154
6.25	Interpolation of luma half-pel positions	155
6.26	Interpolation of luma quarter-pel positions	156
6.27	Luma region interpolated to quarter-pel positions	156
6.28	Interpolation of chroma eighth-pel positions	157
6.29	Macroblock partitions and sub-macroblock partitions	158
6.30	Current and neighbouring partitions : same partition sizes	159
6.31	Current and neighbouring partitions : different partition sizes	159
6.32	Scaled motion vector example	160
6.33	Temporal direct motion vector example	161
6.34	Forming a motion compensated prediction	162
6.35	Biprediction example	163
6.36	MBAFF: prediction from corresponding field	164
6.37	P slice showing partition choices. Reproduced by permission of Elecard	165
6.38	B slice showing macroblock modes. Light-shaded circles are skipped macroblocks. Reproduced by permission of Elecard.	166
6.39	Inter prediction example, P slice	167
6.40	Inter prediction example, B slice	168
6.41	Low delay prediction structure	169

6.42	'Classic' Group of Pictures prediction structure	169
6.43	IPPP. . . with multiple reference pictures	170
6.44	Hierarchical GOP structure	171
6.45	Edge filtering order in a macroblock	172
6.46	Pixels adjacent to vertical and horizontal boundaries	172
6.47	16 × 16 luma macroblock showing block edges	173
6.48	Original frame, 'violin' frame 2	174
6.49	Reconstructed, QP = 36, no filter	175
6.50	Reconstructed, QP = 36, with filter	175
6.51	Reconstructed, QP = 32, no filter	176
6.52	Reconstructed, QP = 32, with filter	176
7.1	Re-scaling and inverse transform	181
7.2	Forward transform and quantization	181
7.3	Luma forward transform: default	182
7.4	Luma inverse transform: default	182
7.5	Luma forward transform: Intra 16 × 16 mode	183
7.6	Luma inverse transform: Intra 16 × 16 mode	183
7.7	Luma forward transform: 8 × 8 transform	183
7.8	Luma inverse transform: 8 × 8 transform	183
7.9	Chroma forward transform: 4:2:0 macroblock	184
7.10	Chroma inverse transform: 4:2:0 macroblock	184
7.11	Chroma forward transform: 4:2:2 macroblock	184
7.12	Chroma inverse transform: 4:2:2 macroblock	184
7.13	Development of the forward transform and quantization process	185
7.14	Development of the rescaling and inverse transform process	186
7.15	Quantization parameter QP vs. effective quantizer step size, logarithmic y-axis	192
7.16	Frequency dependent quantization, 4 × 4 block	204
7.17	Progressive scan orders for 4 × 4 and 8 × 8 blocks	206
7.18	Field scan orders for 4 × 4 and 8 × 8 blocks	207
7.19	CAVLC encoder overview	211
7.20	CABAC coding process overview	218
8.1	Baseline, Constrained Baseline, Extended and Main Profiles	225
8.2	Main and High Profiles	226
8.3	Main and Intra Profiles	227
8.4	Selected Level constraints	228
8.5	Selected display resolutions	229
8.6	H.264 encoder and decoder buffers	230
8.7	Hypothetical Reference Decoder (HRD)	230
8.8	HRD example 1: encoder buffer	232
8.9	HRD example 1: decoder CPB	232
8.10	HRD Example 2: encoder buffer	234
8,.11	HRD Example 2: decoder CPB	234
8.12	HRD Example 3: encoder buffer	235
8.13	HRD Example 3: decoder CPB	236
8.14	Bitstream conformance testing	236

8.15 Decoder conformance testing 237
8.16 Arbitrary Slice Order: Example 238
8.17 FMO: Interleaved map, QCIF, 3 slice groups 239
8.18 FMO: Dispersed macroblock map, QCIF, 4 slice groups 239
8.19 FMO: Foreground and Background map, 4 slice groups 240
8.20 FMO: Box-out, Raster and Wipe maps 240
8.21 Switching streams using I-slices 241
8.22 Switching streams using SP-slices 242
8.23 Encoding SP-slice A_2 (simplified) 242
8.24 Encoding SP-slice B_2 (simplified) 242
8.25 Decoding SP-slice A_2 (simplified) 243
8.26 Encoding SP-slice AB_2 (simplified) 243
8.27 Decoding SP-slice AB_2 244
8.28 Fast-forward using SP-slices 244
8.29 Encapsulation of H.264 syntax elements 245
8.30 MPEG-2 Transport Stream 246
8.31 RTP packet structure (simplified) 246
8.32 ISO Media File 247
8.33 Block diagram from US patent 3679821 (redrawn) 251
8.34 Issued US patents including the terms ‘video coding’ or ‘video compression’, 1990–2007. Source: USPTO patent database. 251
9.1 JM software operation 256
9.2 Planar YCbCr file format, 4:2:0 sampling 257
9.3 JM encoder configuration file 258
9.4 JM encoder output display 258
9.5 Original, reconstructed and decoded frames, container.qcif, QP = 32 259
9.6 Section of coded frame, JM encoder (left), ×264 encoder (right) 262
9.7 Screenshot of an H.264 stream analyzer: Baseline Profile frame. Reproduced by permission of Elecard 263
9.8 Screenshot of stream analyzer: Main Profile frame. Courtesy of Elecard 264
9.9 Foreman/QCIF/Basic complexity 266
9.10 Foreman/QCIF/Basic complexity and options 267
9.11 Foreman/QCIF/Basic and Medium Complexity 268
9.12 Foreman/QCIF/Medium complexity with rate control 269
9.13 Foreman, QCIF sequences: coding time 269
9.14 QCIF sequences: rate vs. PSNR 270
9.15 CIF sequences: rate vs. PSNR 270
9.16 CIF sequences: coding time 271
9.17 Rate-distortion comparison of prediction structures 271
9.18 Sample frames from sequences using different prediction structures, coded at 280kbps 272
9.19 Carphone, QCIF: H.264 vs. MPEG-4 Visual 273
9.20 Frame from ‘Foreman’ sequence showing macroblock sizes 274
9.21 Encoder with rate feedback 275
9.22 Bitrate allocation for rate control 275
9.23 Foreman, QCIF, 100 frames: coded bitrate 278

9.24	Foreman, QCIF, 100 frames: QP per frame	278
9.25	Foreman, QCIF, 100 frames: Luma PSNR per frame	279
9.26	Available macroblock prediction modes	280
9.27	Rate and MSE costs for different coding options	281
9.28	Rate, quality and complexity	283
10.1	Multiple streams/simulcast	288
10.2	Multiple streams/scalable	289
10.3	Overview of scalability types	291
10.4	Temporally scalable sequence, 3 layers	292
10.5	Decoding a temporally scalable sequence	293
10.6	Hierarchical prediction structure	293
10.7	Quality Scalability	295
10.8	Spatial scalability, two layers	296
10.9	Medium Grain Quality Scalability	298
10.10	Combined Spatial, Temporal and Quality scalability	299
10.11	Simulcast vs. scalable bitrates	300
10.12	Quality scalability at CIF resolution	301
10.13	Spatial + temporal scalability, CIF → 4CIF resolution	301
10.14	Spatial scalability, 720p → 1080p resolution	302
10.15	Three views of the same scene	302
10.16	Multiview video: view examples	303
10.17	Multiview video: views and frames	303
10.18	Interview prediction of key frames	304
10.19	Inter-view prediction of all frames	305
10.20	Overview of configurable video codec	306
10.21	Reconfigurable Video Coding scenario	307
10.22	Fully Configurable Video Coding framework	309

List of Tables

2.1 Video frame formats 17
2.2 ITU-R BT.601-5 Parameters 18
2.3 HD display formats 19
3.1 SAE of residual frame after motion compensation, 16×16 block size 37
3.2 Probability of occurrence of motion vectors in sequence 1 59
3.3 Huffman codes for sequence 1 motion vectors 60
3.4 Probability of occurrence of motion vectors in sequence 2 61
3.5 Huffman codes for sequence 2 motion vectors 62
3.6 MPEG-4 Visual Transform Coefficient (TCOEF) VLCs : partial, all codes < 9 bits 63
3.7 MPEG4 Motion Vector Difference (MVD) VLCs 65
3.8 Motion vectors, sequence 1: probabilities and sub-ranges 66
3.9 Residual luminance samples : top-right 8×8 block 75
3.10 DCT coefficients 75
3.11 Quantized coefficients 76
3.12 Variable length coding example 77
3.13 Rescaled coefficients 77
3.14 Decoded residual luminance samples 78
4.1 Overview of the H.264 standard document 92
5.1 Syntax examples: format 99
5.2 H.264 Syntax Sections 102
5.3 Selecting a reference picture to re-map 110
5.4 Initial reference picture list 110
5.5 Final reference picture list 111
5.6 Selected NAL unit types 114
5.7 Sequence Parameter Set example 115
5.8 Picture Parameter Set example 116
5.9 Slice types in H.264 117
5.10 Slice Header, IDR/Intra, Frame 0 118
5.11 Slice Header, Inter, Frame 1 118
5.12 Macroblock types 120
5.13 coded_block_pattern examples. 121
5.14 Type and prediction elements for B macroblocks, excluding Direct or 8×8 partitions 124

5.15 I macroblock, example 1 128
5.16 P macroblock, example 2 129
5.17 P macroblock, example 3 130
5.18 P macroblock, example 4 131
5.19 B macroblock, example 5 132
5.20 B macroblock, example 6 134
5.21 B macroblock, example 7 134
5.22 B macroblock, example 8 135
6.1 Intra prediction types 139
6.2 Choice of prediction mode, most probable mode = 1 149
6.3 Reference picture sources 151
6.4 Reference frames and motion vectors for P and B macroblocks 158
6.5 Layers in hierarchical GOP example 171
7.1 V_{i4} values, 4×4 blocks 190
7.2 Table *v* defined in H.264 standard 190
7.3 Estimated Q_{step} ($4\times$blocks) $= \mathbf{V}_{i4} / \mathbf{Si} \cdot 2^6$), element-by-element division 191
7.4 Tables *v* and *m* 194
7.5 8-point Integer Transform Basis C_{f8} 199
7.6 8-point Integer Transform Basis C_{I8} 200
7.7 Estimated Q_{step} (8×8 blocks) 202
7.8 Scaling matrices 205
7.9 Exp-Golomb Codewords 208
7.10 Mappings to code_num 210
7.11 Choice of look-up table for coeff_token 212
7.12 Thresholds for determining whether to increment *suffixLength* 213
7.13 Binarization of MVD magnitude 219
7.14 Context models for bin 1 219
7.15 Context models 220
8.1 Selected formats, frame rates and levels 229
8.2 HRD example 1: access unit sizes 231
8.3 HRD example 2: frame sizes 233
8.4 HRD example 3: frame sizes 235
8.5 Macroblock allocation map types 239
8.6 Switching from stream A to stream B using SP-slices 242
8.7 SEI messages 249
8.8 Video Usability Information: selected parameters 250
9.1 JM software directory structure 257
9.2 Selected parameter sections in the JM encoder configuration file 260
9.3 Section of trace file showing Sequence Parameter Set 262
9.4 'Low Complexity' and 'Basic' configurations 266
9.5 'Basic' plus options 267
9.6 'Basic', 'Best Baseline', 'Medium' 268
9.7 Bitrate and delay constraints 274
10.1 Example: Quality Scalability 291
10.2 Example: Spatial + Temporal Scalability 292

1

Introduction

1.1 A change of scene

2000:

Most viewers receive analogue television via terrestrial, cable or satellite transmission.

VHS video tapes are the principal medium for recording and playing TV programs, movies, etc.

Cell phones are cell phones, i.e. a mobile handset can only be used to make calls or send SMS messages.

Internet connections are slow, primarily over telephone modems for home users.

Web pages are web pages, with static text, graphics and photos and not much else.

Video calling requires dedicated videoconferencing terminals and expensive leased lines. Video calling over the internet is possible but slow, unreliable and difficult to set up.

Consumer video cameras, camcorders, use tape media, principally analogue tape. Home-made videos generally stay within the home.

2010:

Most viewers receive digital television via terrestrial, cable, satellite or internet, with benefits such as a greater choice of channels, electronic programme guides and high definition services. Analogue TV has been switched off in many countries. Many TV programmes can be watched via the internet.

DVDs are the principal medium for playing pre-recorded movies and TV programs. Many alternatives exist, most of them digital, including internet movie downloading (legal and not-so-legal), hard-disk recording and playback and a

The H.264 Advanced Video Compression Standard, Second Edition Iain E. Richardson
© 2010 John Wiley & Sons, Ltd

variety of digital media formats. High definition DVDs, Blu-Ray Disks, are increasing in popularity.

Cell phones function as cameras, web browsers, email clients, navigation systems, organizers and social networking devices. Occasionally they are used to make calls.

Home internet access speeds continue to get faster via broadband and mobile connections, enabling widespread use of video-based web applications.

Web pages are applications, movie players, games, shopping carts, bank tellers, social networks, etc, with content that changes dynamically.

Video calling over the internet is commonplace with applications such as Skype and iChat. Quality is still variable but continues to improve.

Consumer video cameras use hard disk or flash memory card media. Editing, uploading and internet sharing of home videos is widespread.

A whole range of illegal activities has been born – DVD piracy, movie sharing via the internet, recording and sharing of assaults, etc.

Video footage of breaking news items such as the Chilean earthquake is more likely to come from a cell phone than a TV camera.

All these changes in a ten-year period signify a small revolution in the way we create, share and watch moving images. Many factors have contributed to the shift towards digital video – commercial factors, legislation, social changes and technological advances. From the technology viewpoint, these factors include better communications infrastructure, with widespread, relatively inexpensive access to broadband networks, 3G mobile networks, cheap and effective wireless local networks and higher-capacity carrier transmission systems; increasingly sophisticated devices, with a bewildering array of capabilities packed into a lightweight cellular handset; and the development of easy-to-use applications for recording, editing, sharing and viewing video material. This book will focus on one technical aspect that is key to the widespread adoption of digital video technology – video compression.

Video compression or video encoding is the process of reducing the amount of data required to represent a digital video signal, prior to transmission or storage. The complementary operation, decompression or decoding, recovers a digital video signal from a compressed representation, prior to display. Digital video data tends to take up a large amount of storage or transmission capacity and so video encoding and decoding, or **video coding**, is essential for any application in which storage capacity or transmission bandwidth is constrained. Almost all consumer applications for digital video fall into this category, for example:

- Digital television broadcasting: TV programmes are coded prior to transmission over a limited-bandwidth terrestrial, satellite or cable channel (Figure 1.1).
- Internet video streaming: Video is coded and stored on a server. The coded video is transmitted (streamed) over the internet, decoded on a client and displayed (Figure 1.1).

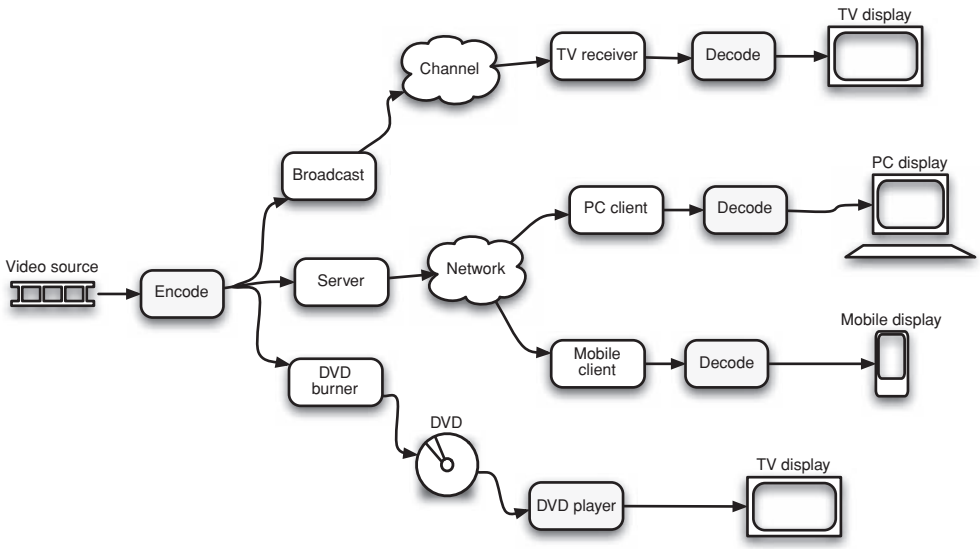


Figure 1.1 Video coding scenarios, one-way

- Mobile video streaming: As above, but the coded video is transmitted over a mobile network such as GPRS or 3G (Figure 1.1).
- DVD video: Source video is coded and stored on a DVD or other storage medium. A DVD player reads the disk and decodes video for display (Figure 1.1).
- Video calling: Each participant includes an encoder and a decoder (Figure 1.2). Video from a camera is encoded and transmitted across a network, decoded and displayed. This occurs in two directions simultaneously.

Each of these examples includes an encoder, which compresses or encodes an input video signal into a coded bitstream, and a decoder, which decompresses or decodes the coded bitstream to produce an output video signal. The encoder or decoder is often built in to a device such as a video camera or a DVD player.

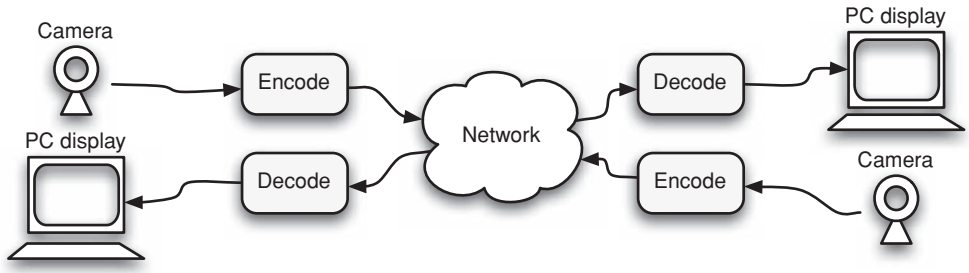


Figure 1.2 Video coding scenario, two-way

1.2 Driving the change

The consumer applications discussed above represent very large markets. The revenues involved in digital TV broadcasting and DVD distribution are substantial. Effective video coding is an essential component of these applications and can make the difference between the success or failure of a business model. A TV broadcasting company that can pack a larger number of high-quality TV channels into the available transmission bandwidth has a market edge over its competitors. Consumers are increasingly discerning about the quality and performance of video-based products and there is therefore a strong incentive for continuous improvement in video coding technology. Even though processor speeds and network bandwidths continue to increase, a better video codec results in a better product and therefore a more competitive product. This drive to improve video compression technology has led to significant investment in video coding research and development over the last 15-20 years and to rapid, continuous advances in the state of the art.

1.3 The role of standards

Many different techniques for video coding have been proposed and researched. Hundreds of research papers are published each year describing new and innovative compression techniques. In contrast to this wide range of innovations, commercial video coding applications tend to use a limited number of standardized techniques for video compression. Standardized video coding formats have a number of potential benefits compared with non-standard, proprietary formats:

- Standards simplify inter-operability between encoders and decoders from different manufacturers. This is important in applications where each 'end' of the system may be produced by a different company, e.g. the company that records a DVD is typically not the same as the company that manufactures a DVD player.
- Standards make it possible to build platforms that incorporate video, in which many different applications such as video codecs, audio codecs, transport protocols, security and rights management, interact in well-defined and consistent ways.
- Many video coding techniques are patented and therefore there is a risk that a particular video codec implementation may infringe patent(s). The techniques and algorithms required to implement a standard are well-defined and the cost of licensing patents that cover these techniques, i.e. licensing the right to use the technology embodied in the patents, can be clearly defined.

Despite recent debates about the benefits of royalty-free codecs versus industry standard video codecs [i], video coding standards are very important to a number of major industries. With the ubiquitous presence of technologies such as DVD/Blu-Ray, digital television, internet video and mobile video, the dominance of video coding standards is set to continue for some time to come.

1.4 Why H.264 Advanced Video Coding is important

This book is about a standard, jointly published by the International Telecommunications Union (ITU) and the International Standards Organisation (ISO) and known by several names:

'H.264', 'MPEG-4 Part 10' and 'Advanced Video Coding'. The standard itself is a document over 550 pages long and filled with highly technical definitions and descriptions. Developed by a team consisting of hundreds of video compression experts, the Joint Video Team, a collaborative effort between the Moving Picture Experts Group (MPEG) and the Video Coding Experts Group (VCEG), this document is the culmination of many man-years' work. It is almost impossible to read and understand without an in-depth knowledge of video coding.

Why write a book about this document? Whilst the standard itself is arguably only accessible to an insider expert, H.264/AVC has huge significance to the broadcast, internet, consumer electronics, mobile and security industries, amongst others. H.264/AVC is the latest in a series of standards published by the ITU and ISO. It describes and defines a method of coding video that can give better performance than any of the preceding standards. H.264 makes it possible to compress video into a smaller space, which means that a compressed video clip takes up less transmission bandwidth and/or less storage space compared to older codecs. A combination of market expansion, technology advances and increased user expectation is driving demand for better, higher quality digital video. For example:

- TV companies are delivering more content in High Definition. Most new television sets can display HD pictures. Customers who pay a premium for High Definition content expect correspondingly high image quality.
- An ever-increasing army of users are uploading and downloading videos using sites such as YouTube. Viewers expect rapid download times and high resolution.
- Recording and sharing videos using mobile handsets is increasingly commonplace.
- Internet video calls, whilst still variable in quality, are easier to make and more widely used than ever.
- The original DVD-Video format, capable of supporting only a single movie in Standard Definition seems increasingly limited.

In each case, better video compression is the key to delivering more, higher-quality video in a cost effective way. H.264 compression makes it possible to transmit HD television over a limited-capacity broadcast channel, to record hours of video on a Flash memory card and to deliver massive numbers of video streams over an already busy internet.

The benefits of H.264/AVC come at a price. The standard is complex and therefore challenging to the engineer or designer who has to develop, program or interface with an H.264 codec. H.264 has more options and parameters – more 'control knobs' – than any previous standard codec. Getting the controls and parameters 'right' for a particular application is not an easy task. Get it right and H.264 will deliver high compression performance; get it wrong and the result is poor-quality pictures and/or poor bandwidth efficiency. Computationally expensive, an H.264 coder can lead to slow coding and decoding times or rapid battery drain on handheld devices. Finally, H.264/AVC, whilst a published industry standard, is not free to use. Commercial implementations are subject to licence fees and the intellectual property position in itself is complicated.

1.5 About this book

The aim of this book is to de-mystify H.264 and its complexities. H.264/AVC will be a key component of the digital media industry for some time to come. A better understanding of

the technology behind the standard and of the inter-relationships of its many component parts should make it possible to get the most out of this powerful tool.

This book is organized as follows.

Chapter 2 explains the concepts of digital video and covers source formats and visual quality measures.

Chapter 3 introduces video compression and the functions found in a typical video codec, such as H.264/AVC and other block-based video compression codecs.

Chapter 4 gives a high-level overview of H.264/AVC at a relatively non-technical level.

Chapters 5, 6 and 7 cover the standard itself in detail. **Chapter** 5 deals with the H.264/AVC syntax, i.e. the construction of an H.264 bitstream) including picture formats and picture management. **Chapter 6** describes the prediction methods supported by the standard, intra and inter prediction. **Chapter 7** explains the residual coding processes, i.e. transform and quantization and symbol coding.

Chapter 8 deals with issues closely related to the main standard – storage and network transport of H.264 data, conformance or how to ensure compatibility with H.264 and licensing, including the background and details of the intellectual property licence associated with H.264 implementations.

Chapter 9 examines the implementation and performance of H.264. It explains how to experiment with H.264, the effect of H.264 parameters on performance, implementation challenges and performance optimization.

Chapter 10 covers extensions to H.264/AVC, in particular the Scalable and Multiview Video Coding extensions that have been published since the completion of the H.264 standard. It examines possible future developments, including Reconfigurable Video Coding, a more flexible way of specifying and implementing video codecs, and possible successors to H.264, currently being examined by the standards groups.

Readers of my earlier book, "H.264 and MPEG-4 Video Compression", may be interested to know that Chapters 4–10 are largely or completely new material.

1.6 Reference

i. Ian Hickson, 'Codecs for <audio> and <video>', HTML5 specification discussion, http://lists.whatwg.org/htdig.cgi/whatwg-whatwg.org/2009-June/020620.html, accessed August 2009.

2

Video formats and quality

2.1 Introduction

Video coding is the process of compressing and decompressing a digital video signal. This chapter examines the structure and characteristics of digital images and video signals and introduces concepts such as sampling formats and quality metrics. Digital video is a representation of a natural or real-world visual scene, sampled spatially and temporally. A scene is typically sampled at a point in time to produce a frame, which represents the complete visual scene at that point in time, or a field, which typically consists of odd- or even-numbered lines of spatial samples. Sampling is repeated at intervals (e.g. 1/25 or 1/30 second intervals) to produce a moving video signal. Three components or sets of samples are typically required to represent a scene in colour. Popular formats for representing video in digital form include the ITU-R 601 standard, High Definition formats and a set of 'intermediate formats'. The accuracy of a reproduction of a visual scene must be measured to determine the performance of a visual communication system, a notoriously difficult and inexact process. Subjective measurements are time consuming and prone to variations in the response of human viewers. Objective or automatic measurements are easier to implement but as yet do not accurately match the behaviour of a human observer.

2.2 Natural video scenes

A 'real world' or natural video scene is typically composed of multiple objects each with their own characteristic shape, depth, texture and illumination. The colour and brightness of a natural video scene changes with varying degrees of smoothness throughout the scene, i.e. it has continuous tone. Characteristics of a typical natural video scene (Figure 2.1) that are relevant for video processing and compression include spatial characteristics such as texture variation within scene, number and shape of objects, colour, etc, and temporal characteristics such as object motion, changes in illumination and movement of the camera or viewpoint.

The H.264 Advanced Video Compression Standard, Second Edition Iain E. Richardson
© 2010 John Wiley & Sons, Ltd

Figure 2.1 Still image from natural video scene

2.3 Capture

A natural visual scene is spatially and temporally continuous. Representing a visual scene in digital form involves sampling the real scene spatially, usually on a rectangular grid in the video image plane, and temporally, as a series of still frames or components of frames sampled at regular intervals in time (Figure 2.2). Digital video is the representation of a sampled video scene in digital form. Each spatio-temporal sample, a picture element or pixel, is represented as one or more numbers that describes the brightness or luminance and the colour of the sample.

To obtain a 2-D sampled image, a camera focuses a 2-D projection of the video scene onto a sensor, such as an array of Charge Coupled Devices (CCDs). In the case of colour image capture, each colour component is separately filtered and projected onto a CCD array (see section 2.4).

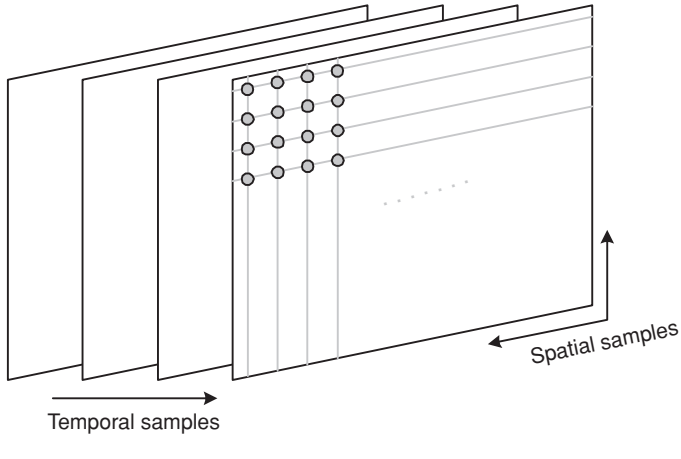


Figure 2.2 Spatial and temporal sampling of a video sequence

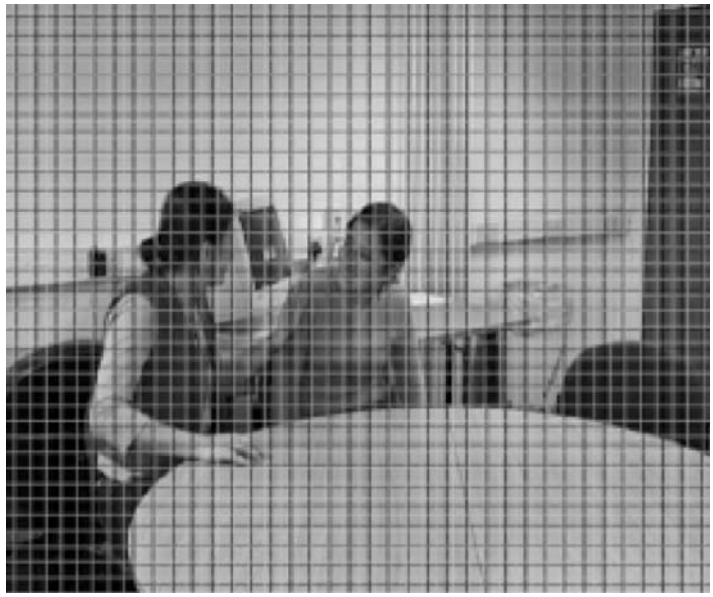


Figure 2.3 Image with two sampling grids

2.3.1 Spatial sampling

The output of a CCD array is an analogue video signal, a varying electrical signal that represents a video image. Sampling the signal at a point in time produces a sampled image or frame that has defined values at a set of sampling points. The most common format for a sampled image is a rectangle with the sampling points positioned on a square or rectangular grid. Figure 2.3 shows a continuous-tone frame with two different sampling grids superimposed upon it. Sampling occurs at each of the intersection points on the grid and the sampled image may be reconstructed by representing each sample as a square picture element or pixel. The number of sampling points influences the visual quality of the image. Choosing a 'coarse' sampling grid, the black grid in Figure 2.3, produces a low-resolution sampled image (Figure 2.4) whilst increasing the number of sampling points slightly, the grey grid in Figure 2.3, increases the resolution of the sampled image (Figure 2.5).

2.3.2 Temporal sampling

A moving video image is formed by taking a rectangular 'snapshot' of the signal at periodic time intervals. Playing back the series of snapshots or frames produces the appearance of motion. A higher temporal sampling rate or frame rate gives apparently smoother motion in the video scene but requires more samples to be captured and stored. Frame rates below 10 frames per second may be used for very low bit-rate video communications, because the amount of data is relatively small, but motion is clearly jerky and unnatural at this rate. Between 10–20 frames per second is more typical for low bit-rate video communications; the image is smoother but jerky motion may be visible in fast-moving parts of the sequence. Temporal

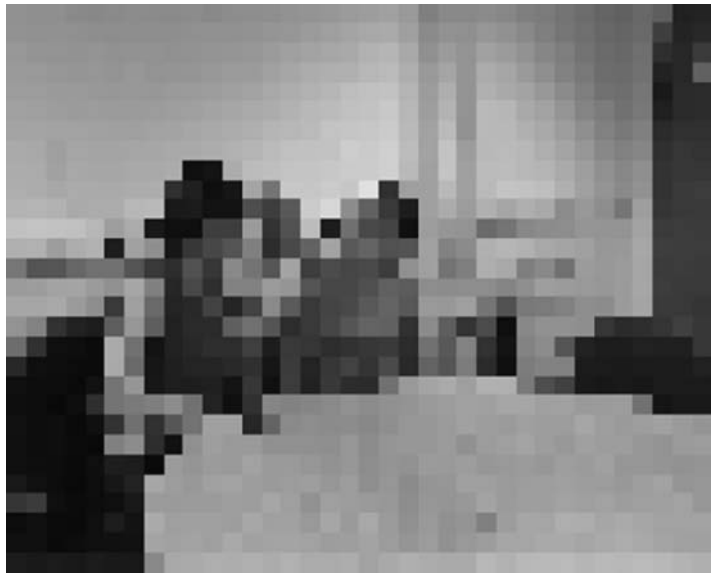


Figure 2.4 Image sampled at coarse resolution (black sampling grid)

Figure 2.5 Image sampled at finer resolution (grey sampling grid)

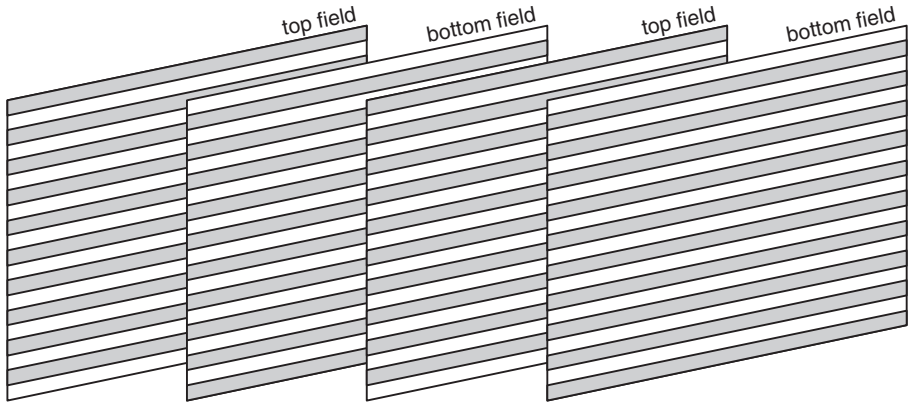


Figure 2.6 Interlaced video sequence

sampling at 25 or 30 complete frames per second is the norm for Standard Definition television pictures, with interlacing to improve the appearance of motion, see below; 50 or 60 frames per second produces very smooth apparent motion at the expense of a very high data rate.

2.3.3 Frames and fields

A video signal may be sampled as a series of complete frames, **progressive** sampling, or as a sequence of interlaced fields, **interlaced** sampling. In an interlaced video sequence, half of the data in a frame, one field, is typically sampled at each temporal sampling interval. A field may consist of either the odd-numbered or even-numbered lines within a complete video frame and an interlaced video sequence (Figure 2.6) typically contains a series of fields, each representing half of the information in a complete video frame, illustrated in Figure 2.7 and Figure 2.8. The

Figure 2.7 Top field

Figure 2.8 Bottom field

advantage of this sampling method is that it is possible to send twice as many fields per second as the number of frames in an equivalent progressive sequence with the same data rate, giving the appearance of smoother motion. For example, a PAL video sequence consists of 50 fields per second and when played back, motion appears smoother than in an equivalent progressive video sequence containing 25 frames per second. Increasingly, video content may be captured and/or displayed in progressive format. When video is captured in one format (e.g. interlaced) and displayed in another (e.g. progressive), it is necessary to convert between formats.

2.4 Colour spaces

Most digital video applications rely on the display of colour video and so need a mechanism to capture and represent colour information. A monochrome image (Figure 2.1) requires just one number to indicate the brightness or luminance of each spatial sample. Colour images, on the other hand, require at least three numbers per pixel position to accurately represent colour. The method chosen to represent brightness, luminance or luma and colour is described as a **colour space**.

2.4.1 RGB

In the RGB colour space, a colour image sample is represented with three numbers that indicate the relative proportions of Red, Green and Blue, the three additive primary colours of light. Combining red, green and blue in varying proportions can create any colour. Figure 2.9 shows the red, green and blue components of a colour image: the red component consists of all the red samples, the green component contains all the green samples and the blue component contains the blue samples. The person on the right is wearing a blue sweater and so this appears ‘brighter’ in the blue component, whereas the red waistcoat of the figure on the left

Figure 2.9 Red, Green and Blue components of colour image

appears brighter in the red component. The RGB colour space is well suited to capture and display of colour images. Capturing an RGB image involves filtering out the red, green and blue components of the scene and capturing each with a separate sensor array. Colour displays show an RGB image by separately illuminating the red, green and blue components of each pixel according to the intensity of each component. From a normal viewing distance, the separate components merge to give the appearance of 'true' colour.

2.4.2 YCrCb

The human visual system (HVS) is less sensitive to colour than to luminance. In the RGB colour space the three colours are equally important and so are usually all stored at the same resolution but it is possible to represent a colour image more efficiently by separating the luminance from the colour information and representing luma with a higher resolution than colour.

The Y:Cr:Cb colour space is a popular way of efficiently representing colour images. Y is the luminance component and can be calculated as a weighted average of R, G and B:

$$Y = k_r R + k_g G + k_b B \tag{2.1}$$

where k are weighting factors.

The colour information can be represented as **colour difference** (chrominance or chroma) components, where each chrominance component is the difference between R, G or B and the luminance Y:

$$\begin{aligned} Cr &= R - Y \\ Cb &= B - Y \\ Cg &= G - Y \end{aligned} \tag{2.2}$$

The complete description of a colour image is given by Y, the luminance component, and three colour differences Cr, Cb and Cg that represent the difference between the colour intensity and the mean luminance of each image sample. Figure 2.10 shows the red, green and blue chroma components corresponding to the RGB components of Figure 2.9. Here, mid-grey is zero difference, light grey is a positive difference and dark grey is a negative difference. The chroma components only have significant values where there is a large difference between the

Figure 2.10 Cr, Cg and Cb components

colour component and the luma image (Figure 2.1). Note the strong blue and red difference components.

So far, this representation has little obvious merit since we now have four components instead of the three in RGB. However, Cr+Cb+Cg is a constant and so only two of the three chrominance components need to be stored or transmitted since the third component can always be calculated from the other two. In the Y:Cr:Cb colour space, only the luma (Y) and red and blue chroma (Cr, Cb) are transmitted. Y:Cr:Cb has an important advantage over RGB, in that the Cr and Cb components may be represented with a **lower resolution** than Y because the HVS is less sensitive to colour than luminance. This reduces the amount of data required to represent the chrominance components without having an obvious effect on visual quality. To the casual observer, there is no obvious difference between an RGB image and a Y:Cr:Cb image with reduced chrominance resolution. Representing chroma with a lower resolution than luma in this way is a simple but effective form of image compression.

An RGB image may be converted to Y:Cr:Cb after capture in order to reduce storage and/or transmission requirements. Before displaying the image, it is usually necessary to convert back to RGB. The equations for converting an RGB image to and from Y:Cr:Cb colour space and vice versa are given in (2.3 and 2.4). Note that G can be extracted from the Y:Cr:Cb representation by subtracting Cr and Cb from Y, demonstrating that it is not necessary to store or transmit a Cg component.

$$\begin{aligned} Y &= 0.299R + 0.587G + 0.114B \\ Cb &= 0.564(B - Y) \\ Cr &= 0.713(R - Y) \end{aligned} \tag{2.3}$$

$$\begin{aligned} R &= Y + 1.402Cr \\ G &= Y - 0.344Cb - 0.714Cr \\ B &= Y + 1.772Cb \end{aligned} \tag{2.4}$$

2.4.3 YCrCb sampling formats

Figure 2.11 shows three sampling patterns for Y, Cr and Cb that are supported by H.264/AVC. 4:4:4sampling means that the three components (Y:Cr:Cb) have the same resolution and hence

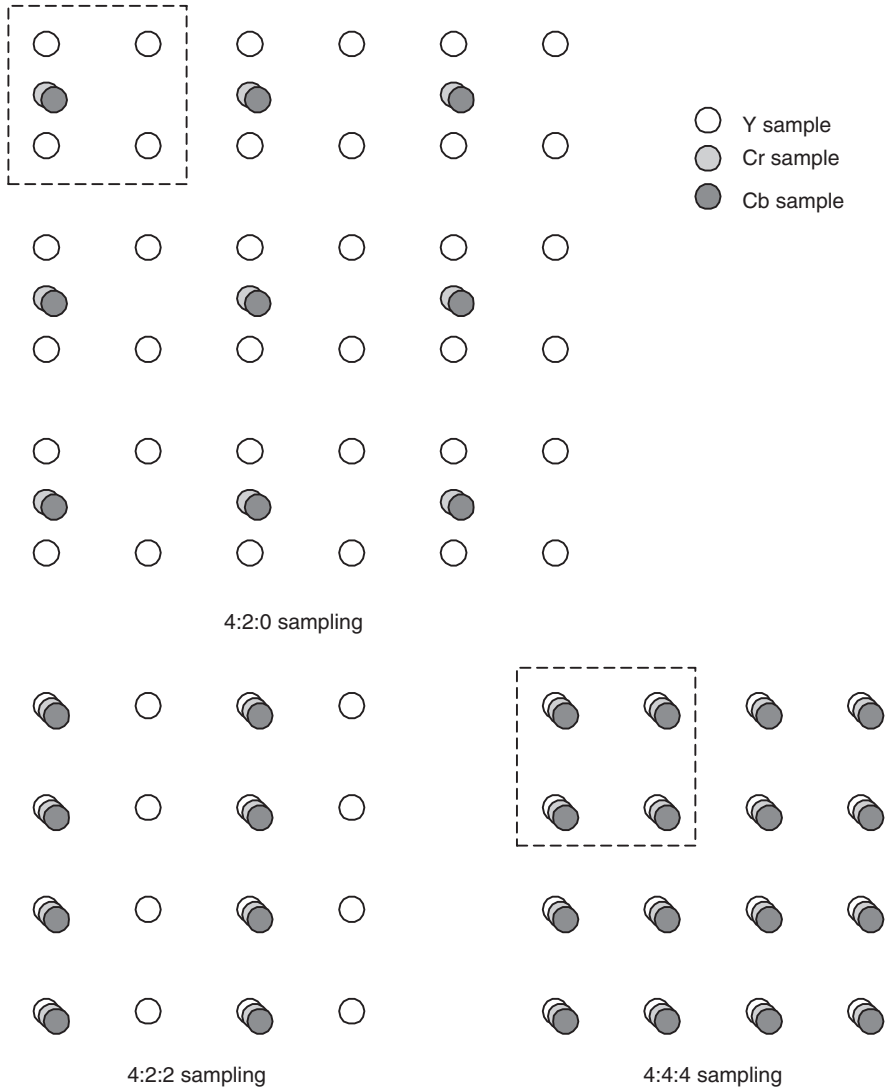


Figure 2.11 4:2:0, 4:2:2 and 4:4:4 sampling patterns (progressive)

a sample of each component exists at every pixel position. The numbers indicate the relative sampling rate of each component in the **horizontal** direction, i.e. for every 4 luminance samples there are 4 Cr and 4 Cb samples. 4:4:4 sampling preserves the full fidelity of the chrominance components. In 4:2:2 sampling, sometimes referred to as YUY2, the chrominance components have the same vertical resolution as the luma but half the horizontal resolution. The numbers 4:2:2 mean that for every 4 luminance samples in the horizontal direction there are 2 Cr and 2 Cb samples. 4:2:2 video is used for high-quality colour reproduction.

In the popular 4:2:0 sampling format ('YV12'), Cr and Cb each have half the horizontal and vertical resolution of Y. The term '4:2:0' is rather confusing because the numbers do not actually have a logical interpretation and appear to have been chosen historically as a 'code' to identify this particular sampling pattern and to differentiate it from 4:4:4 and 4:2:2. 4:2:0 sampling is widely used for consumer applications such as video conferencing, digital television and digital versatile disk (DVD) storage. Because each colour difference component contains ¼ of the number of samples in the Y component, 4:2:0 Y:Cr:Cb video requires exactly ½ as many samples as 4:4:4 or R:G:B video.

Example

Image resolution: 720×576 pixels
Y resolution: 720×576 samples, each represented with 8 bits
4:4:4 Cr, Cb resolution: 720×576 samples, each 8 bits
Total number of bits: $720 \times 576 \times 8 \times 3 = 9953280$ bits
4:2:0 Cr, Cb resolution: 360×288 samples, each 8 bits
Total number of bits: $(720 \times 576 \times 8) + (360 \times 288 \times 8 \times 2) = 4976640$ bits
The 4:2:0 version requires half as many bits as the 4:4:4 version.

4:2:0 sampling is sometimes described as '12 bits per pixel'. The reason for this can be seen by examining a group of 4 pixels, enclosed in dotted lines in Figure 2.11. Using 4:4:4 sampling, a total of 12 samples are required, 4 each of Y, Cr and Cb, requiring a total of $12 \times 8 = 96$ bits, an average of $96/4 = 24$ bits per pixel. Using 4:2:0 sampling, only 6 samples are required, 4 Y and one each of Cr, Cb, requiring a total of $6 \times 8 = 48$ bits, an average of $48/4 = 12$ bits per pixel.

In a 4:2:0 interlaced video sequence, the Y, Cr and Cb samples corresponding to a complete video frame are allocated to two fields.

Figure 2.12 shows the method of allocating Y, Cr and Cb samples to a pair of fields adopted in H.264. It is clear from this figure that the total number of samples in a pair of fields is the same as the number of samples in an equivalent progressive frame.

2.5 Video formats

2.5.1 Intermediate formats

The video compression algorithms described in this book can compress a wide variety of video frame formats. In practice, it is common to capture or convert to one of a set of 'intermediate formats' prior to compression and transmission. The Common Intermediate Format (CIF) is the basis for a popular set of formats listed in Table 2.1. Figure 2.13 shows the luma component of a video frame sampled at a range of resolutions, from 4CIF down to Sub-QCIF. The choice of frame resolution depends on the application and available storage or transmission capacity. For example, 4CIF is appropriate for standard-definition television and DVD-video; CIF and QCIF are popular for videoconferencing applications; QCIF or SQCIF are appropriate for mobile multimedia applications where the display resolution and the bitrate are limited.

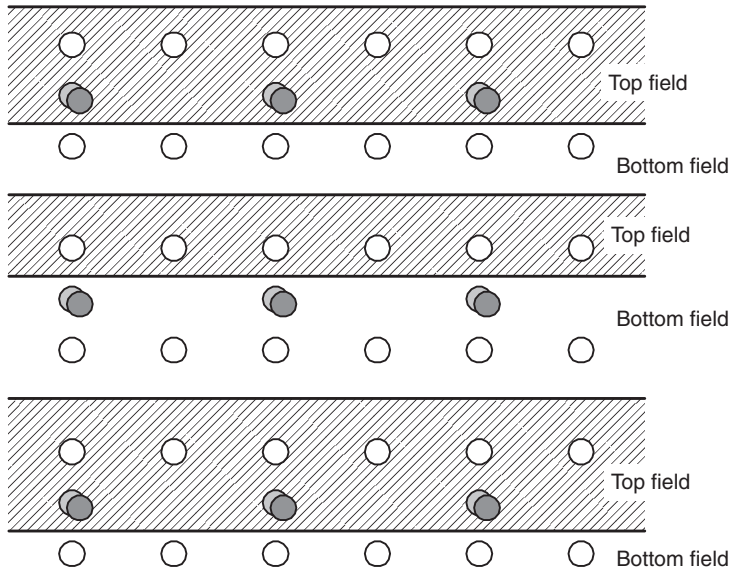


Figure 2.12 Allocation of 4:2:0 samples to top and bottom fields

Table 2.1 lists the number of bits required to represent 1 uncompressed frame in each format, assuming 4:2:0 sampling and 8 bits per luma and chroma sample.

2.5.2 Standard Definition

A widely-used format for digitally coding video signals for television production is ITU-R Recommendation BT.601-5 [i]. Note that the term 'coding' in the Recommendation title means conversion to digital format and does not imply compression. The luminance component of the video signal is sampled at 13.5MHz and the chrominance at 6.75MHz to produce a 4:2:2 Y:Cr:Cb component signal. The parameters of the sampled digital signal depend on the video frame rate, 30Hz for an NTSC signal and 25Hz for a PAL/SECAM signal, and are shown in Table 2.2. The higher 30Hz frame rate of NTSC is compensated for by a lower spatial resolution so that the total bit rate is the same in each case, 216Mbps. The actual area shown

Table 2.1 Video frame formats

Format	Luminance resolution (horiz. × vert.)	Bits per frame (4:2:0, 8 bits per sample)
Sub-QCIF	128 × 96	147456
Quarter CIF (QCIF)	176 × 144	304128
CIF	352 × 288	1216512
4CIF	704 × 576	4866048

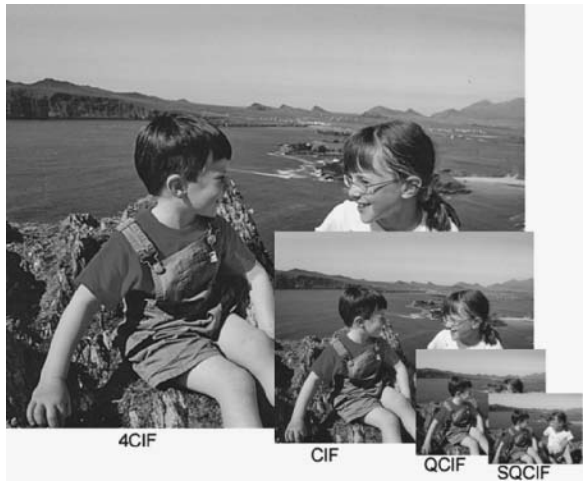


Figure 2.13 Video frame sampled at range of resolutions

on the display, the **active area**, is smaller than the total because it excludes horizontal and vertical blanking intervals that exist 'outside' the edges of the frame.

Each sample has a possible range of 0 to 255. Levels of 0 and 255 are reserved for synchronization and the active luminance signal is restricted to a range of 16 (black) to 235 (white).

2.5.3 High Definition

Several High Definition (HD) video formats exist [ii]. The most widely used television display formats are listed in Table 2.3 and shown graphically in Figure 2.14. Note that equivalent versions are defined for base frame rates of 30Hz, rather than the European 25Hz base frame

Table 2.2 ITU-R BT.601-5 Parameters

	30Hz frame rate	25Hz frame rate
Fields per second	60	50
Lines per complete frame	525	625
Luminance samples per line	858	864
Chrominance samples per line	429	432
Bits per sample	8	8
Total bit rate	216 Mbps	216 Mbps
Active lines per frame	480	576
Active samples per line (Y)	720	720
Active samples per line (Cr,Cb)	360	360

Table 2.3 HD display formats

Format	Progressive or Interlaced	Horizontal pixels	Vertical pixels	Frames or fields per second
720p	Progressive	1280	720	25 frames
1080i	Interlaced	1920	1080	50 fields
1080p	Progressive	1920	1080	25 frames

rate. It is clear that HD formats require an even larger uncompressed storage or transmission rate than SD formats. SD video has $(720 \times 576 \times 25) = 10368000$ displayed pixels per second. 720p HD video has $(1280 \times 720 \times 25) = 23040000$ displayed pixels per second and at the top end, 1080p HD has $(1920 \times 1080 \times 25) = 51840000$ displayed pixels per second. The very large storage or transmission capacity required for uncompressed video at these resolutions means it is essential to compress video for practical applications.

2.6 Quality

In order to specify, evaluate and compare video communication systems it is necessary to determine the quality of the video images displayed to the viewer. Measuring visual quality is a difficult and often imprecise art because there are so many factors that can affect the results. Visual quality is inherently **subjective** and is therefore influenced by many subjective factors that make it difficult to obtain a completely accurate measure of quality. For example, a viewer's opinion of visual quality can depend very much on the task at hand, such as passively watching a DVD movie, actively participating in a videoconference or trying to identify a person in

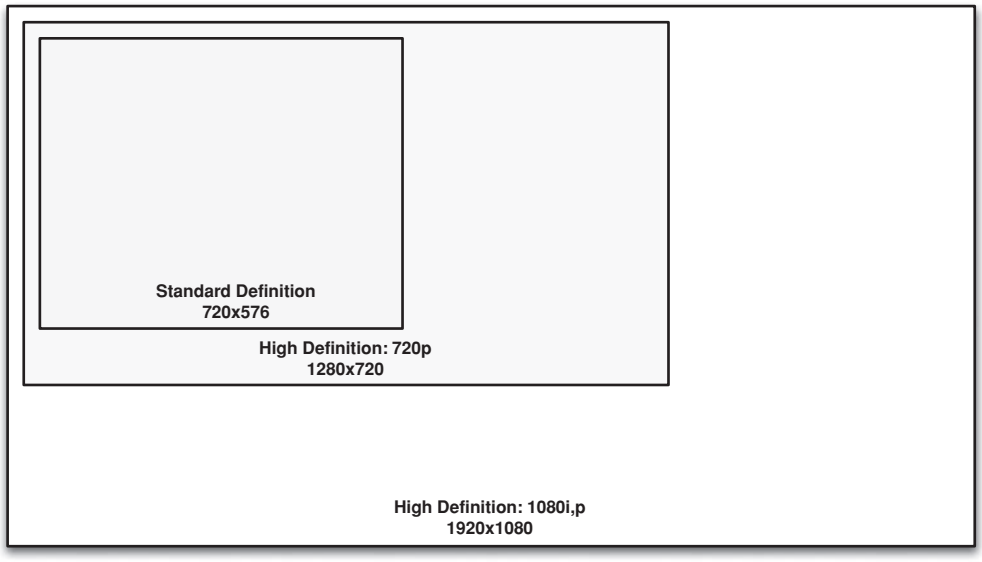


Figure 2.14 SD and HD formats

a surveillance video scene. Measuring visual quality using **objective** criteria gives accurate, repeatable results but as yet there are no objective measurement systems that completely reproduce the subjective experience of a human observer watching a video display.

2.6.1 Subjective quality measurement

2.6.1.1 Factors influencing subjective quality

Our perception of a visual scene is formed by a complex interaction between the components of the Human Visual System (HVS), the eye and the brain. The perception of visual quality is influenced by spatial fidelity, i.e. how clearly parts of the scene can be seen, whether there is any obvious distortion, and temporal fidelity, i.e. whether motion appears natural and 'smooth'. However, a viewer's opinion of 'quality' is also affected by other factors such as the viewing environment, the observer's state of mind and the extent to which the observer interacts with the visual scene. A user carrying out a specific task that requires concentration on part of a visual scene will have a quite different requirement for quality than a user who is passively watching a movie. It has been shown that a viewer's opinion of visual quality is measurably higher if the viewing environment is comfortable and non-distracting, regardless of the quality of the visual image itself.

Other important influences on perceived quality include visual attention, i.e. the way an observer perceives a scene by fixating on a sequence of points in the image rather than by taking in everything simultaneously [iii], and the so-called 'recency effect', i.e. our opinion of a visual sequence is more heavily influenced by recently-viewed material than older video material [iv, v]. All of these factors make it very difficult to accurately and quantitatively measure visual quality.

2.6.1.2 ITU-R 500

Several test procedures for subjective quality evaluation are defined in ITU-R Recommendation BT.500-11 [vi]. A widely-used procedure from the standard is the Double Stimulus Continuous Quality Scale (DSCQS) method in which an assessor is presented with a pair of images or short video sequences A and B, one after the other, and is asked to give A and B a 'quality score' by marking on a continuous line with five intervals ranging from 'Excellent' to 'Bad'. In a typical test session, the assessor is shown a series of pairs of sequences and is asked to grade each pair. Within each pair of sequences, one is an unimpaired 'reference' sequence and the other is the same sequence, modified by a system or process under test. Figure 2.15 shows an experimental set-up appropriate for the testing of a video CODEC in which the original sequence is compared with the same sequence after encoding and decoding. The selection of which sequence is 'A' and which is 'B' is randomized.

The order of the two sequences, original and 'impaired', is randomized during the test session so that the assessor does not know which is the original and which is the impaired sequence. This helps prevent the assessor from pre-judging the impaired sequence compared with the reference sequence. At the end of the session, the scores are converted to a normalized range and the end result is a score, sometimes described as a 'mean opinion score' (MOS) that indicates the **relative** quality of the impaired and reference sequences.

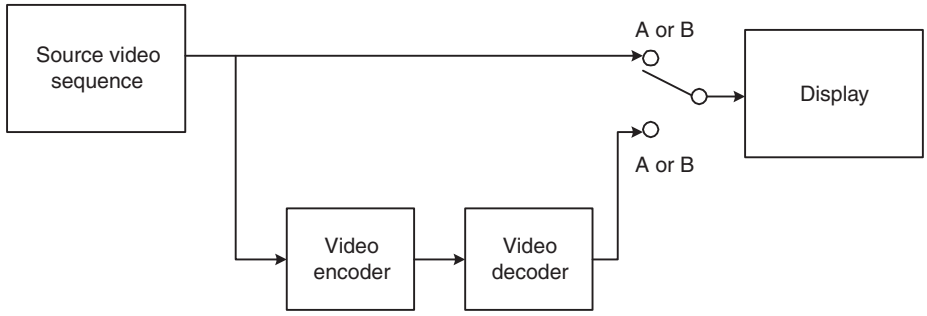


Figure 2.15 DSCQS testing system

Tests such as DSCQS are accepted as realistic measures of subjective visual quality. However, this type of test suffers from practical problems. The results can vary significantly depending on the assessor and also on the video sequence under test. This variation is compensated for by repeating the test with several sequences and several assessors. An 'expert' assessor who is familiar with the nature of video compression distortions or 'artefacts' may give a biased score and it is recommended to use 'non-expert' assessors. This means that a large pool of assessors is required because a non-expert assessor will quickly learn to recognize characteristic artefacts in the video sequences and so will become 'expert'. These factors make it expensive and time consuming to carry out the DSCQS tests thoroughly.

2.6.2 Objective quality measurement

The complexity and cost of subjective quality measurement make it attractive to be able to measure quality automatically using an algorithm. Developers of video compression and video processing systems rely heavily on so-called objective (algorithmic) quality measures. The most widely used measure is Peak Signal to Noise Ratio (PSNR) but the limitations of this metric have led to many efforts to develop more sophisticated measures that approximate the response of 'real' human observers.

2.6.2.1 PSNR

Peak Signal to Noise Ratio (PSNR) (2.5) is measured on a logarithmic scale and depends on the mean squared error (MSE) between an original and an impaired image or video frame, relative to $(2^n - 1)^2$, the square of the highest-possible signal value in the image, where n is the number of bits per image sample.

$$PSNR_{dB} = 10 \log_{10} \frac{(2^n - 1)^2}{MSE} \tag{2.5}$$

PSNR can be calculated easily and quickly and is therefore a very popular quality measure, widely used to compare the 'quality' of compressed and decompressed video images. Figure 2.16 shows a close-up of three images: the first image (a) is the original and (b) and (c) are degraded (blurred) versions of the original image. Image (b) has a measured PSNR of 30.6dB whilst image (c) has a PSNR of 28.3dB, reflecting the poorer image quality.

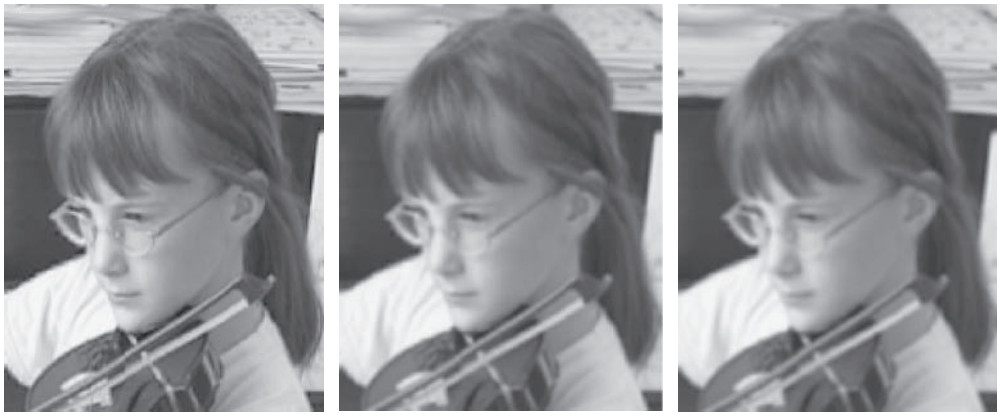


Figure 2.16 PSNR examples: (a) Original; (b) 30.6dB; (c) 28.3dB

The PSNR metric suffers from a number of limitations. PSNR requires an unimpaired original image for comparison but this may not be available in every case and it may not be easy to verify that an 'original' image has perfect fidelity. PSNR does not correlate well with subjective video quality measures such as ITU-R 500. For a given image or image sequence, high PSNR usually indicates high quality and low PSNR usually indicates low quality. However, a particular value of PSNR does not necessarily equate to an 'absolute' subjective quality. For example, Figure 2.17 shows a distorted version of the original image from Figure 2.16 in which only the background of the image has been blurred. This image has a PSNR of 27.7dB relative to the original. Most viewers would rate this image as significantly better than image (c) in Figure 2.16 because the face is clearer, in opposition to the PSNR rating. This example shows that PSNR ratings do not necessarily correlate with 'true' subjective

Figure 2.17 Image with blurred background (PSNR = 27.7dB)

quality. In this case, a human observer gives a higher importance to the face region and so is particularly sensitive to distortion in this area.

2.6.2.2 Other objective quality metrics

Because of the limitations of crude metrics such as PSNR, there has been a lot of work in recent years to try to develop a more sophisticated objective test that more closely approaches subjective test results. Many different approaches have been proposed. It has proved difficult to establish reliable objective video quality metrics that accurately predict the results of subjective tests. Recent proposals have included Just Noticeable Difference (JND) [vii], Digital Video Quality (DVQ) [viii], Structual SIMilarity index (SSIM) [ix], PSNRplus [x] and Predicted Mean Opinion Score (MOSp) [xi]. These metrics have varying degrees of success in predicting subjective test scores, with reported correlations of between 70 per cent and 90 per cent between each objective metric and measured subjective quality scores.

The ITU-T Video Quality Experts Group (VQEG) aims to develop industry standards related to video and multimedia quality assessment. VQEG has developed Recommendation J.247, which covers 'full reference' video quality measurement, i.e. quality metrics that require access to an original, uncompressed version of a video signal [xii]. The Recommendation lists four objective quality metrics that 'can be recommended by ITU-T at this time':

A: NTT Full Reference Method.

B: OPTICOM Perceptual Video Quality Method.

C: Psytechnics Full Reference Method.

D: Yonsei Full Reference Method.

The general approach of these methods is as follows. First, the original (reference) and test (degraded) video sequences are aligned spatially and temporally. Next, a series of degradation parameters are calculated. Each of the four metrics calculates a different set of parameters such as blurring, edges introduced by compression, blockiness, etc. Finally, these parameters are combined to produce a single number that is an estimate of subjective quality. The correlation between subjective and estimated quality is reported as ranging from 77 per cent to 84 per cent in the Recommendation, indicating that there is still scope for developing better Full Reference objective quality metrics.

An even more challenging task is to measure or estimate quality when a full reference, an unimpaired copy of the original video, is not available. This is the case in many practical applications. For example, the original source may not be available in the case of user-generated video content, or it may be desirable to monitor quality at the receiver, for example in a customer's digital television receiver, where there is no access to the original video. In these situations, No Reference (NR) or Reduced Reference (RR) quality estimation is required. No Reference metrics attempt to estimate subjective quality based only on characteristics of the decoded video clip. This is a difficult task but some success has been reported using methods such as modelling typical image/video compression artefacts [xiii, xiv]. Reduced Reference metrics calculate a quality 'signature', typically a low-bitrate side signal, which is

communicated to the decoder. A quality estimate is formed by processing the decoded video clip together with the side information [xv].

2.7 Summary

Sampling analogue video produces a digital video signal, which has the advantages of accuracy, quality and compatibility with digital media and transmission but which typically occupies a prohibitively large bitrate. Issues inherent in digital video systems include spatial and temporal resolution, colour representation and the measurement of visual quality. The next chapter introduces the basic concepts of video compression, necessary to accommodate digital video signals on practical storage and transmission media.

2.8 References

i. ITU-R Recommendation BT.601-6, 'Studio encoding parameters of digital television for standard 4:3 and widescreen 16:9 aspect ratios', ITU-R, 2007.
ii. 'High Definition Image Formats for Television Broadcasting', European Broadcasting Union (EBU) Technical Report 3299, Geneva, 2004.
iii. J. Findlay and I. Gilchrist, *Active Vision: the psychology of looking and seeing*. Oxford University Press, 2003.
iv. N. Wade and M. Swanston, (2001) *Visual Perception: An Introduction*. 2nd edition. London: Psychology Press.
v. R. Aldridge, J. Davidoff, D. Hands, M. Ghanbari and D. E. Pearson, 'Recency effect in the subjective assessment of digitally coded television pictures', proc. Fifth International Conference on Image Processing and its applications, Heriot-Watt University, Edinburgh, UK (July 1995).
vi. ITU-T Recommendation BT.500-11, 'Methodology for the subjective assessment of the quality of television pictures', ITU-T, 2002.
vii. J. Lubin and D. Fibush, 'Sarnoff JND vision model', T1A1.5 Working group Document, T1 Standards Committee, 1997.
viii. A.B. Watson, J. Hu and J.F. McGowan III, 'Digital video quality metric based on human vision', *Journal of Electronic imaging*, vol. 10, no.1, Jan 2001, pp. 20–29.
ix. Z. Wang and A.C. Bovik, 'A universal image quality index', IEEE Signal Proc. Letters, vol. 9, no. 3, Mar. 2002, pp. 81–84.
x. T. Oelbaum, K Diepold and W. Zia, 'A generic method to increase the prediction accuracy of visual quality metrics', PCS 2007.
xi. A. Bhat, I.E. Richardson and C.S. Kannangara, 'A Novel Perceptual Quality Metric for Video Compression,' Proc. International Picture Coding Symposium 2009, Chicago, May 2009.
xii. ITU-T Recommendation J.247, 'Objective perceptual multimedia video quality measurement in the presence of a full reference', ITU-T, August 2008.
xiii. Z. Wang, H. Sheikh and A. Bovik, 'No-reference perceptual quality assessment of JPEG compressed images', Proc. International Conference on Image Processing, 2002.
xiv. R. Dosselmann and X. Yang, 'A prototype no-reference video quality system', Proc. Canadian Conference on Computer and Robot Vision, 2007.
xv. Z. Wang and E. Simoncelli, 'Reduced-reference image quality assessment using a wavelet-domain natural image statistic model', Proc. SPIE Human Vision and Electronic Imaging, Vol. 5666, 2005.

3

Video coding concepts

3.1 Introduction

compress *vb.*: to squeeze together or compact into less space; condense

compression *noun*: the act of compression or the condition of being compressed

Compression is the act or process of compacting data into a smaller number of bits. Video compression (video coding) is the process of converting digital video into a format suitable for transmission or storage, whilst typically reducing the number of bits. 'Raw' or uncompressed digital video typically requires a large bitrate, approximately 216Mbits for 1 second of uncompressed Standard Definition video, see Chapter 2, and compression is necessary for practical storage and transmission of digital video.

Compression involves a complementary pair of systems, a compressor (encoder) and a decompressor (decoder). The encoder converts the source data into a compressed form occupying a reduced number of bits, prior to transmission or storage, and the decoder converts the compressed form back into a representation of the original video data. The encoder/decoder pair is often described as a **CODEC** (en**CO**der/**DEC**oder) (Figure 3.1).

Data compression is achieved by removing **redundancy,** i.e. components that are not necessary for faithful reproduction of the data. Many types of data contain **statistical** redundancy and can be effectively compressed using **lossless** compression, so that the reconstructed data at the output of the decoder is a perfect copy of the original data. Unfortunately, lossless compression of image and video information gives only a moderate amount of compression. The best that can be achieved with lossless image compression standards such as JPEG-LS [i] is a compression ratio of around 3–4 times. **Lossy** compression is necessary to achieve higher compression. In a lossy compression system, the decompressed data is not identical to the source data and much higher compression ratios can be achieved at the expense of a loss of visual quality. Lossy video compression systems are based on the principle of removing **subjective** redundancy, elements of the image or video sequence that can be removed without significantly affecting the viewer's perception of visual quality.

Most video coding methods exploit both **temporal** and **spatial** redundancy to achieve compression. In the temporal domain, there is usually a high correlation or similarity between

The H.264 Advanced Video Compression Standard, Second Edition Iain E. Richardson
© 2010 John Wiley & Sons, Ltd

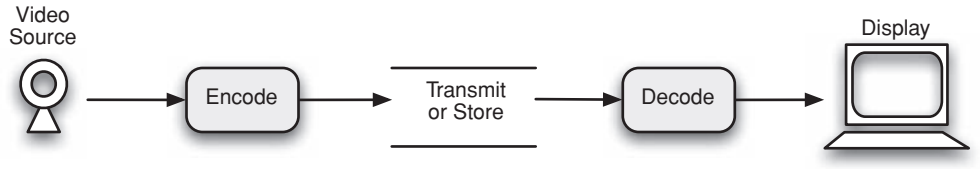


Figure 3.1 Encoder / Decoder

frames of video that were captured at around the same time. Temporally adjacent frames, i.e. successive frames in time order, are often highly correlated, especially if the temporal sampling rate or frame rate is high. In the spatial domain, there is usually a high correlation between pixels (samples) that are close to each other, i.e. the values of neighbouring samples are often very similar (Figure 3.2).

The H.264 Advanced Video Coding standard shares a number of common features with other popular compression formats such as MPEG-2 Video, MPEG-4 Visual, H.263, VC-1, etc. Each of these formats is based upon a CODEC 'model' that uses prediction and/or block-based motion compensation, transform, quantization and entropy coding. In this chapter we examine the main components of this model, starting with the prediction model, including intra prediction, motion estimation and compensation, and continuing with image transforms, quantization, predictive coding and entropy coding. The chapter concludes with a 'walk-through' of the basic model, following through the process of encoding and decoding a block of image samples.

3.2 Video CODEC

A video CODEC (Figure 3.3) encodes a source image or video sequence into a compressed form and decodes this to produce a copy or approximation of the source sequence. If the

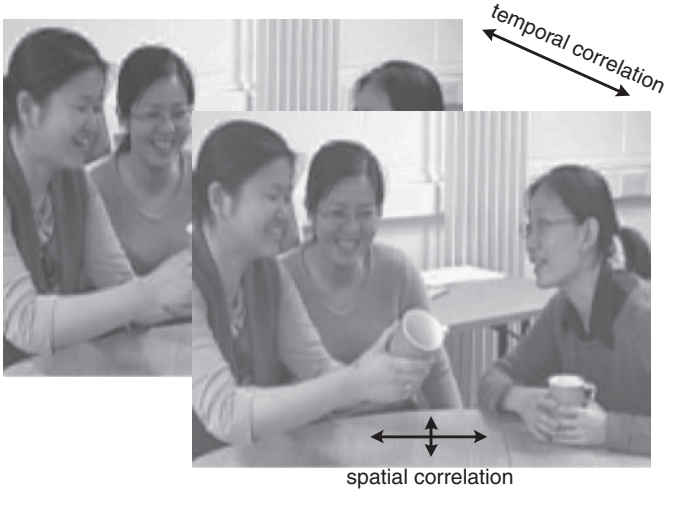


Figure 3.2 Spatial and temporal correlation in a video sequence

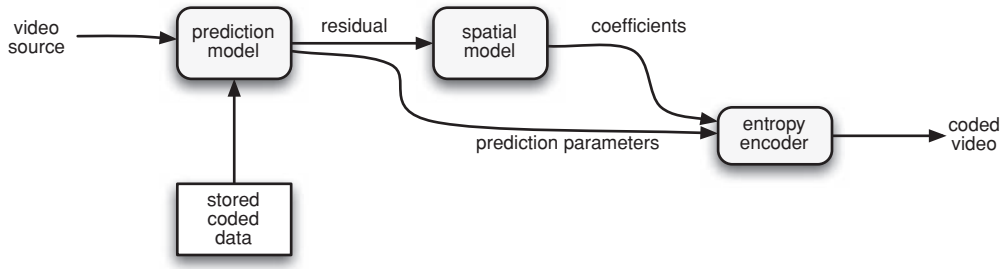


Figure 3.3 Video encoder block diagram

decoded video sequence is identical to the original, then the coding process is lossless; if the decoded sequence differs from the original, the process is lossy.

The CODEC represents the original video sequence by a **model**, an efficient coded representation that can be used to reconstruct an approximation of the video data. Ideally, the model should represent the sequence using as few bits as possible and with as high a fidelity as possible. These two goals, compression efficiency and high quality, are usually conflicting, i.e. a lower compressed bit rate typically produces reduced image quality at the decoder.

A video encoder (Figure 3.3) consists of three main functional units: a **prediction model**, a **spatial model** and an **entropy encoder**. The input to the prediction model is an uncompressed 'raw' video sequence. The prediction model attempts to reduce redundancy by exploiting the similarities between neighbouring video frames and/or neighbouring image samples, typically by constructing a prediction of the current video frame or block of video data. In H.264/AVC, the prediction is formed from data in the current frame or in one or more previous and/or future frames. It is created by spatial extrapolation from neighbouring image samples, **intra prediction**, or by compensating for differences between the frames, **inter** or **motion compensated prediction**. The output of the prediction model is a residual frame, created by subtracting the prediction from the actual current frame, and a set of model parameters indicating the intra prediction type or describing how the motion was compensated.

The residual frame forms the input to the spatial model which makes use of similarities between local samples in the residual frame to reduce spatial redundancy. In H.264/AVC this is carried out by applying a transform to the residual samples and quantizing the results. The transform converts the samples into another domain in which they are represented by transform coefficients. The coefficients are quantized to remove insignificant values, leaving a small number of significant coefficients that provide a more compact representation of the residual frame. The output of the spatial model is a set of quantized transform coefficients.

The parameters of the prediction model, i.e. intra prediction mode(s) or inter prediction mode(s) and motion vectors, and the spatial model, i.e. coefficients, are compressed by the entropy encoder. This removes statistical redundancy in the data, for example representing commonly occurring vectors and coefficients by short binary codes. The entropy encoder produces a compressed bit stream or file that may be transmitted and/or stored. A compressed sequence consists of coded prediction parameters, coded residual coefficients and header information.

The video decoder reconstructs a video frame from the compressed bit stream. The coefficients and prediction parameters are decoded by an entropy decoder after which the spatial model is decoded to reconstruct a version of the residual frame. The decoder uses the prediction

parameters, together with previously decoded image pixels, to create a prediction of the current frame and the frame itself is reconstructed by adding the residual frame to this prediction.

3.3 Prediction model

The data to be processed are a set of image samples in the current frame or field. The goal of the prediction model is to reduce redundancy by forming a prediction of the data and subtracting this prediction from the current data. The prediction may be formed from previously coded frames (a **temporal prediction**) or from previously coded image samples in the same frame (a **spatial prediction**). The output of this process is a set of residual or difference samples and the more accurate the prediction process, the less energy is contained in the residual. The residual is encoded and sent to the decoder which re-creates the same prediction so that it can add the decoded residual and reconstruct the current frame. In order that the decoder can create an identical prediction, it is essential that the encoder forms the prediction using only data available to the decoder, i.e. data that has already been coded and transmitted.

3.3.1 Temporal prediction

The predicted frame is created from one or more past or future frames known as **reference frames**. The accuracy of the prediction can usually be improved by compensating for motion between the reference frame(s) and the current frame.

3.3.1.1 Prediction from the previous video frame

The simplest method of temporal prediction is to use the previous frame as the predictor for the current frame. Two successive frames from a video sequence are shown in Figure 3.4 and Figure 3.5. Frame 1 is used as a predictor for frame 2 and the residual formed by subtracting the predictor (frame 1) from the current frame (frame 2) is shown in Figure 3.6. In this image, mid-grey represents a difference of zero and light or dark greys correspond to positive and negative differences respectively. The obvious problem with this simple prediction is that a lot of energy remains in the residual frame (indicated by the light and dark areas) and this means that there is still a significant amount of information to compress after temporal prediction. Much of the residual energy is due to object movements between the two frames and a better prediction may be formed by **compensating** for motion between the two frames.

3.3.1.2 Changes due to motion

Causes of changes between video frames include motion, uncovered regions and lighting changes. Types of motion include rigid object motion, for example a moving car, deformable object motion, for example a person speaking, and camera motion such as panning, tilt, zoom and rotation. An uncovered region may be a portion of the scene background uncovered by a moving object. With the exception of uncovered regions and lighting changes, these differences correspond to pixel movements between frames. It is possible to estimate the trajectory of each pixel between successive video frames, producing a field of pixel trajectories known as the

Figure 3.4 Frame 1

Figure 3.5 Frame 2

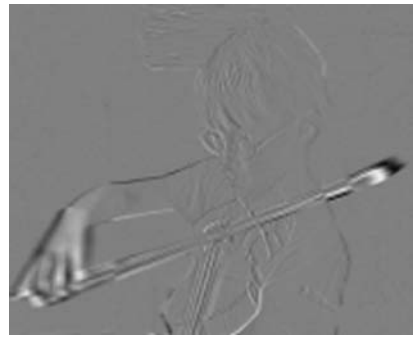


Figure 3.6 Difference

optical flow or optic flow [ii]. Figure 3.7 shows the optical flow field for the frames of Figure 3.4 and Figure 3.5. The complete field contains a flow vector for every pixel position but for clarity, the field is sub-sampled so that only the vector for every 2nd pixel is shown.

If the optical flow field is accurately known, it should be possible to form an accurate prediction of most of the pixels of the current frame by moving each pixel from the reference frame along its optical flow vector. However, this is not a practical method of motion compensation for several reasons. An accurate calculation of optical flow is very computationally intensive,

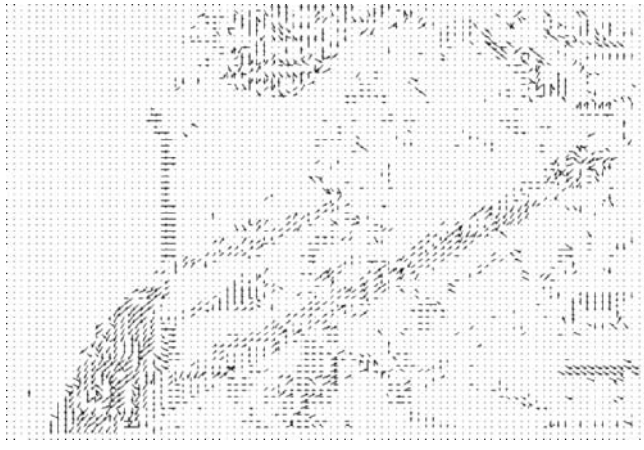


Figure 3.7 Optical flow

since the more accurate methods use an iterative procedure for every pixel, and for the decoder to re-create the prediction frame it would be necessary to send the optical flow vector for every pixel to the decoder resulting in a large amount of transmitted data and negating the advantage of a small residual.

3.3.1.3 Block-based motion estimation and compensation

A practical and widely used method of motion compensation is to compensate for movement of rectangular sections or 'blocks' of the current frame. The following procedure is carried out for each block of MxN samples in the current frame:

- Search an area in the reference frame, a past or future frame, to find a similar MxN-sample region. This search may be carried out by comparing the MxN block in the current frame with some or all of the possible MxN regions in a search area, e.g. a region centred on the current block position, and finding the region that gives the 'best' match. A popular matching criterion is the energy in the residual formed by subtracting the candidate region from the current MxN block, so that the candidate region that minimises the residual energy is chosen as the best match. This process of finding the best match is known as **motion estimation**.
- The chosen candidate region becomes the predictor for the current MxN block (a motion compensated prediction) and is subtracted from the current block to form a residual MxN block (**motion compensation**).
- The residual block is encoded and transmitted and the offset between the current block and the position of the candidate region (**motion vector**) is also transmitted.

The decoder uses the received motion vector to re-create the predictor region. It decodes the residual block, adds it to the predictor and reconstructs a version of the original block.

Block-based motion compensation is popular for a number of reasons. It is relatively straightforward and computationally tractable, it fits well with rectangular video frames and with block-based image transforms such as the Discrete Cosine Transform and it provides a

reasonably effective temporal model for many video sequences. There are however a number of disadvantages. For example, 'real' objects rarely have neat edges that match rectangular boundaries, objects often move by a fractional number of pixel positions between frames and many types of object motion are hard to compensate for using block-based methods, e.g. deformable objects, rotation, warping and complex motion such as a cloud of smoke. Despite these disadvantages, block-based motion compensation is the basis of the temporal prediction model used by all current video coding standards.

3.3.1.4 Motion compensated prediction of a macroblock

The **macroblock**, corresponding to a 16 × 16-pixel region of a frame, is the basic unit for motion compensated prediction in a number of important visual coding standards including MPEG-1, MPEG-2, MPEG-4 Visual, H.261, H.263 and H.264. For source video material in the popular 4:2:0 format (Chapter 2), a macroblock is organised as shown in Figure 3.8. A 16 × 16-pixel region of the source frame is represented by 256 luminance samples arranged in four 8 × 8-sample blocks, 64 red chrominance samples in one 8 × 8 block and 64 blue chrominance samples in one 8 × 8 block, giving a total of six 8 × 8 blocks. An H.264/AVC codec processes each video frame in units of a macroblock.

Motion estimation:
Motion estimation of a macroblock involves finding a 16 × 16-sample region in a reference frame that closely matches the current macroblock. The reference frame is a previously encoded frame from the sequence and may be before or after the current frame in display order. A search area in the reference frame centred on the current macroblock position is searched and the 16 × 16 region within the search area that minimizes a matching criterion is chosen as the 'best match' (Figure 3.9).

Motion compensation:
The luma and chroma samples of the selected 'best' matching region in the reference frame is subtracted from the current macroblock to produce a residual macroblock that is encoded and transmitted together with a motion vector describing the position of the best matching region relative to the current macroblock position.

There are many variations on the basic motion estimation and compensation process. The reference frame may be a previous frame in temporal order, a future frame or a combination

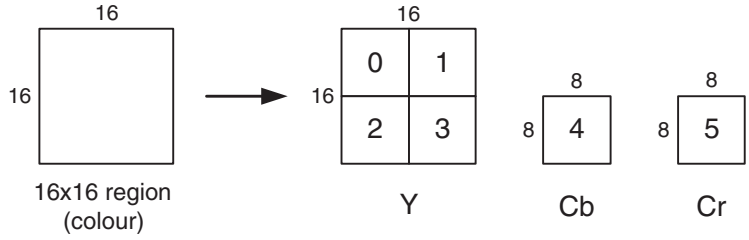


Figure 3.8 Macroblock (4:2:0)

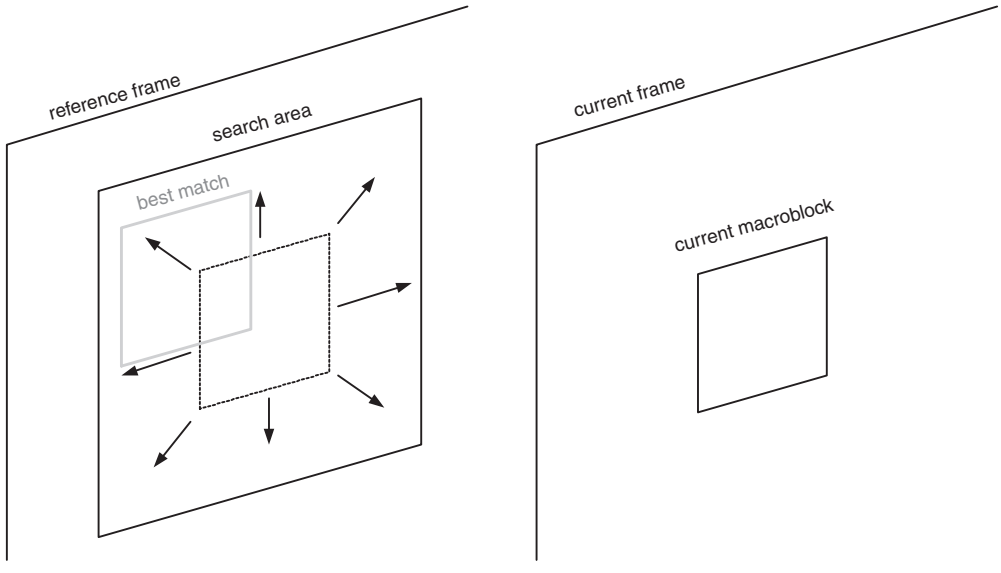


Figure 3.9 Motion estimation

of predictions from two or more previously encoded frames. If a future frame is chosen as the reference, it is necessary to encode this frame before the current frame, i.e. frames must be encoded out of order. Where there is a significant change between the reference and current frames, for example a scene change or an uncovered area, it may be more efficient to encode the macroblock without motion compensation and so an encoder may choose **intra** mode encoding using intra prediction or **inter** mode encoding with motion compensated prediction for each macroblock. Moving objects in a video scene rarely follow 'neat' 16 × 16-pixel boundaries and so it may be more efficient to use a variable block size for motion estimation and compensation. Objects may move by a fractional number of pixels between frames, e.g. 2.78 pixels rather than 2.0 pixels in the horizontal direction, and a better prediction may be formed by interpolating the reference frame to sub-pixel positions before searching these positions for the best match.

3.3.1.5 Motion compensation block size

Two successive frames of a video sequence are shown in Figure 3.10 and Figure 3.11. Frame 1 is subtracted from frame 2 without motion compensation to produce a residual frame (Figure 3.12). The energy in the residual is reduced by motion compensating each 16 × 16 macroblock (Figure 3.13). Motion compensating each 8 × 8 block instead of each 16 × 16 macroblock reduces the residual energy further (Figure 3.14) and motion compensating each 4 × 4 block gives the smallest residual energy of all (Figure 3.15). These examples show that smaller motion compensation block sizes can produce better motion compensation results. However, a smaller block size leads to increased complexity, with more search operations to be carried out, and an increase in the number of motion vectors that need to be transmitted. Sending each

Figure 3.10 Frame 1

Figure 3.11 Frame 2

Figure 3.12 Residual : no motion compensation

Figure 3.13 Residual : 16 × 16 block size

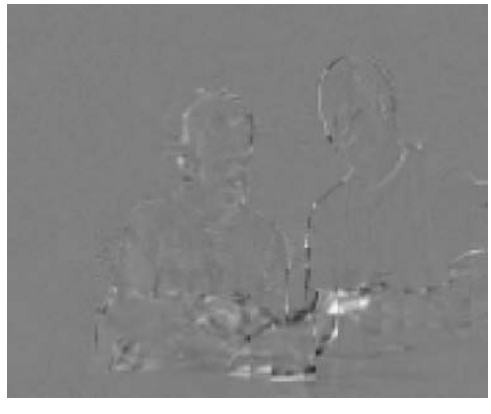


Figure 3.14 Residual : 8 × 8 block size

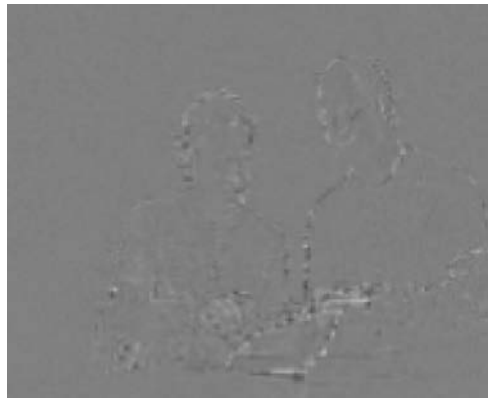


Figure 3.15 Residual : 4 × 4 block size

motion vector requires bits to be transmitted and the extra overhead for vectors may outweigh the benefit of reduced residual energy. An effective compromise is to adapt the block size to the picture characteristics, for example choosing a large block size in flat, homogeneous regions of a frame and choosing a small block size around areas of high detail and complex motion (Chapter 6).

3.3.1.6 Sub-pixel motion compensation

Figure 3.16 shows a close-up view of part of a reference frame. In some cases, predicting from interpolated sample positions in the reference frame may form a better motion compensated prediction. In Figure 3.17, the reference region pixels are interpolated to half-pixel positions and it may be possible to find a better match for the current macroblock by searching the interpolated samples. Sub-pixel motion estimation and compensation involves searching sub-pixel interpolated positions as well as integer-pixel positions and choosing the position that gives the best match and minimizes the residual energy. Figure 3.18 shows the concept of quarter-pixel motion estimation. In the first stage, motion estimation finds the best match on the integer pixel grid (circles). The encoder searches the half-pixel positions immediately next to this best match (squares) to see whether the match can be improved and if required, the quarter-pixel positions next to the best half-pixel position (triangles) are then searched. The final match, at an integer, half-pixel or quarter-pixel position, is subtracted from the current block or macroblock.

The residual in Figure 3.19 is produced using a block size of 4×4 pixels using half-pixel interpolation and has less residual energy than Figure 3.15. This approach may be extended further by interpolation onto a $^1/_4$-pixel grid to give a still smaller residual (Figure 3.20). In

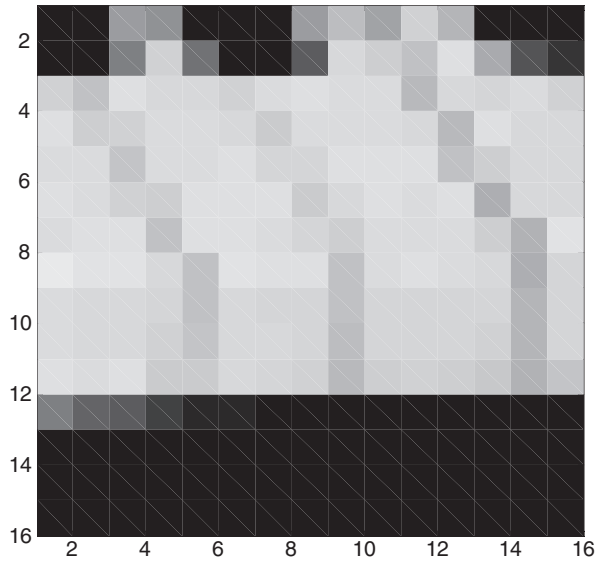


Figure 3.16 Close-up of reference region

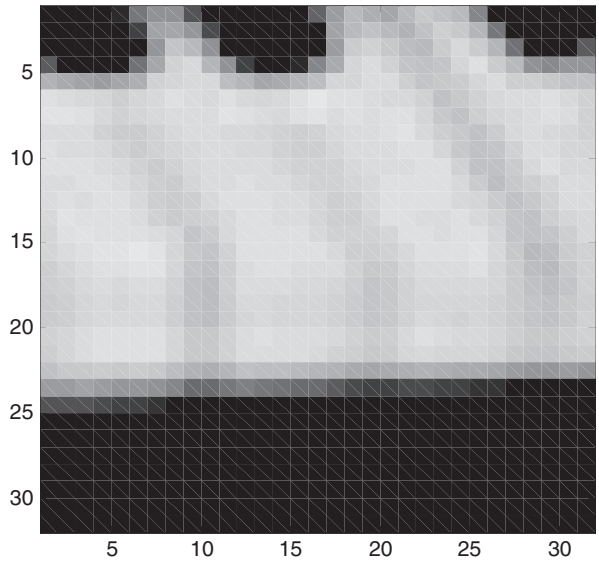


Figure 3.17 Reference region interpolated to half-pixel positions

general, 'finer' interpolation provides better motion compensation performance, producing a smaller residual at the expense of increased complexity. The performance gain tends to diminish as the interpolation steps increase. Half-pixel interpolation gives a significant gain over integer-pixel motion compensation, quarter-pixel interpolation gives a moderate further improvement, eighth-pixel interpolation gives a small further improvement again and so on.

Some examples of the performance achieved by sub-pixel interpolation are given in Table 3.1. A motion-compensated reference frame, the previous frame in the sequence, is subtracted from the current frame and the energy of the residual, approximated by the Sum of Absolute Errors (SAE), is listed in the table. A lower SAE indicates better motion compensation performance. In each case, sub-pixel motion compensation gives improved performance compared with integer-pixel compensation. The improvement from integer to half-pixel is

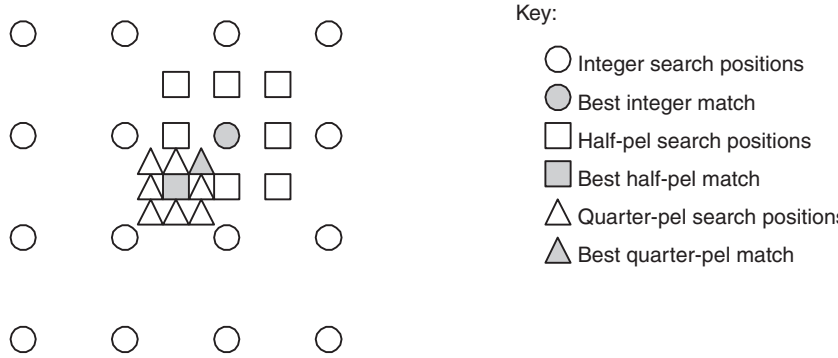


Figure 3.18 Integer, half-pixel and quarter-pixel motion estimation

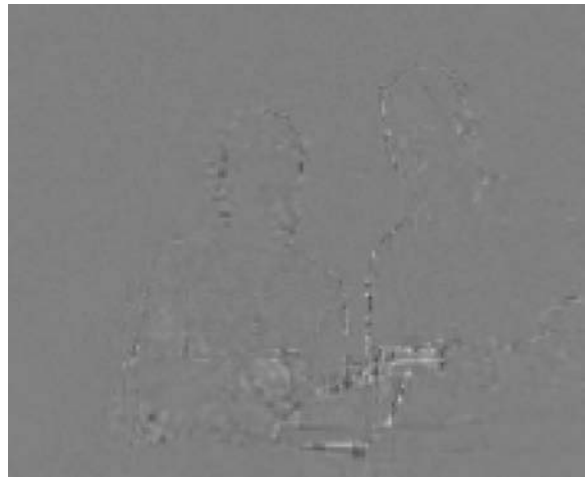


Figure 3.19 Residual : 4 × 4 blocks, 1/2-pixel compensation

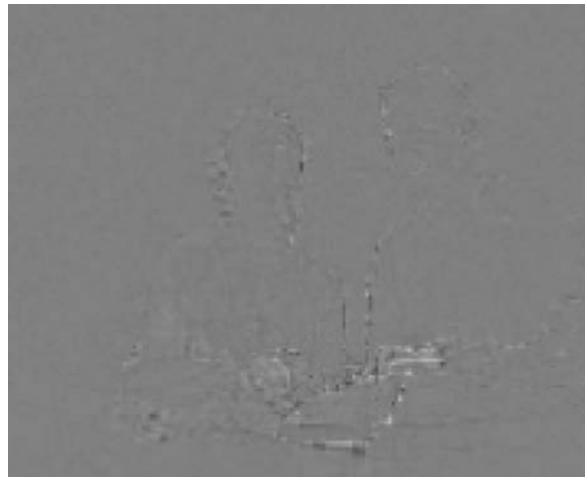


Figure 3.20 Residual : 4 × 4 blocks, 1/4-pixel compensation

Table 3.1 SAE of residual frame after motion compensation, 16 × 16 block size

Sequence	No motion compensation	Integer-pel	Half-pel	Quarter-pel
'Violin', QCIF	171945	153475	128320	113744
'Grasses', QCIF	248316	245784	228952	215585
'Carphone', QCIF	102418	73952	56492	47780

more significant than the further improvement from half- to quarter-pixel. The sequence 'Grasses' has highly complex motion and is particularly difficult to motion-compensate, hence the large SAE; 'Violin' and 'Carphone' are less complex and motion compensation produces smaller SAE values.

Searching for matching 4 × 4 blocks with $^1/_4$-pixel interpolation is considerably more complex than searching for 16 × 16 blocks with no interpolation. In addition to the extra complexity, there is a coding penalty since the vector for every block must be encoded and transmitted to the receiver in order to correctly reconstruct the image. As the block size is reduced, the number of vectors that have to be transmitted increases. More bits are required to represent $\frac{1}{2}$ or $^1/_4$-pixel vectors because the fractional part of the vector, e.g. 0.25 or 0.5, must be encoded as well as the integer part. Figure 3.21 plots the integer motion vectors that are transmitted along with the residual of Figure 3.13. The motion vectors required for the residual of Figure 3.20 are plotted in Figure 3.22, in which there are 16 times as many vectors, each represented by two fractional numbers DX and DY with $^1/_4$-pixel accuracy. There is therefore a trade-off in compression efficiency associated with more complex motion compensation schemes, since more accurate motion compensation requires more bits to encode the vector field, but fewer bits to encode the residual whereas less accurate motion compensation requires fewer bits for the vector field but more bits for the residual. The efficiency of sub-pixel interpolation schemes can be improved by using sophisticated interpolation filters [iii].

3.3.2 Spatial model: intra prediction

The prediction for the current block of image samples is created from previously-coded samples in the same frame. Figure 3.23 shows a block that is to be predicted, centre, in the current frame. Assuming that the blocks of image samples are coded in raster-scan order, which is not always the case, the upper/left shaded blocks are available for intra prediction. These blocks have already been coded and placed in the output bitstream. When the decoder processes the current block, the shaded upper/left blocks are already decoded and can be used to re-create the prediction.

Many different approaches to intra prediction have been proposed. H.264/AVC uses spatial extrapolation to create an intra prediction for a block or macroblock. Figure 3.24 shows the

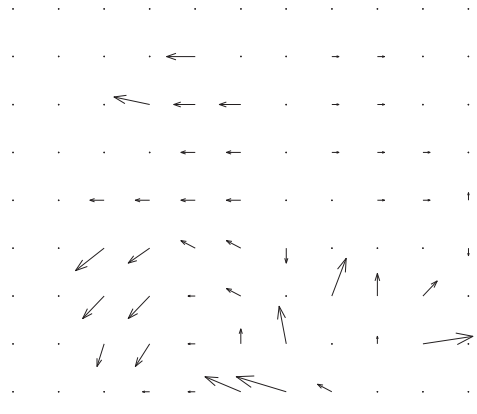


Figure 3.21 Motion vector map : 16 × 16 blocks, integer vectors

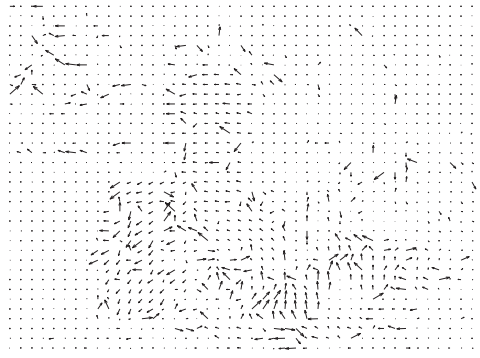


Figure 3.22 Motion vector map : 4 × 4 blocks, 1/4-pixel vectors

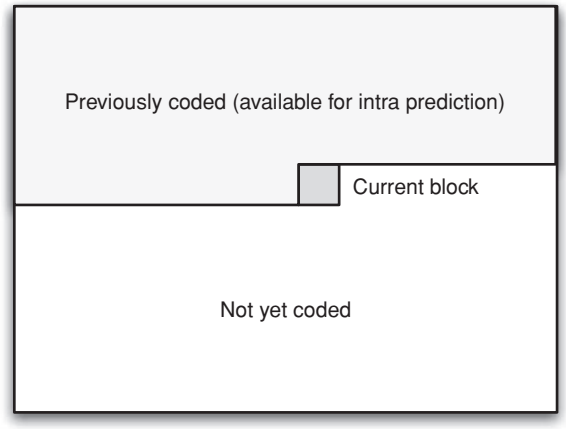


Figure 3.23 Intra prediction: available samples

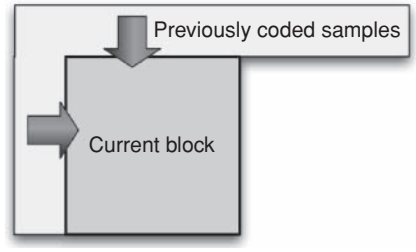


Figure 3.24 Intra prediction: spatial extrapolation

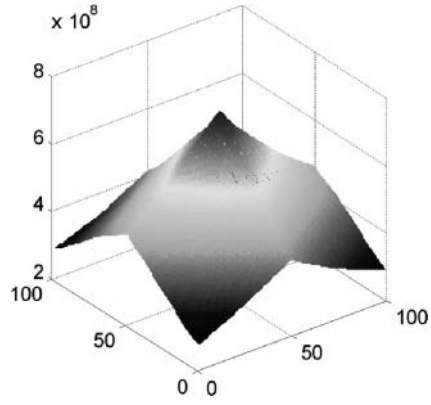


Figure 3.25 2D autocorrelation function of image

general concept. One or more prediction(s) are formed by extrapolating samples from the top and/or left sides of the current block. In general, the nearest samples are most likely to be highly correlated with the samples in the current block (Figure 3.25) and so only the pixels along the top and/or left edges are used to create the prediction block. Once the prediction has been generated, it is subtracted from the current block to form a residual in a similar way to inter prediction. The residual is transformed and encoded, together with an indication of how the prediction was generated. Intra prediction is described in detail in Chapter 6.

3.4 Image model

A natural video image consists of a grid of sample values. Natural images are often difficult to compress in their original form because of the high correlation between neighbouring image samples. Figure 3.25 shows the two-dimensional autocorrelation function of a natural video image (Figure 3.4) in which the height of the graph at each position indicates the similarity between the original image and a spatially-shifted copy of itself. The peak at the centre of the figure corresponds to zero shift. As the spatially-shifted copy is moved away from the original image in any direction, the function drops off as shown in the figure, with the gradual slope indicating that image samples within a local neighbourhood are highly correlated.

A motion-compensated residual image such as Figure 3.20 has an autocorrelation function (Figure 3.26) that drops off rapidly as the spatial shift increases, indicating that neighbouring samples are weakly correlated. Efficient motion compensation or intra prediction reduces local correlation in the residual making it easier to compress than the original video frame. The function of the **image model** is to further decorrelate image or residual data and to convert it into a form that can be efficiently compressed using an entropy coder. Practical image models typically have three main components, transformation to decorrelate and compact the data, quantization to reduce the precision of the transformed data and reordering to arrange the data to group together significant values.

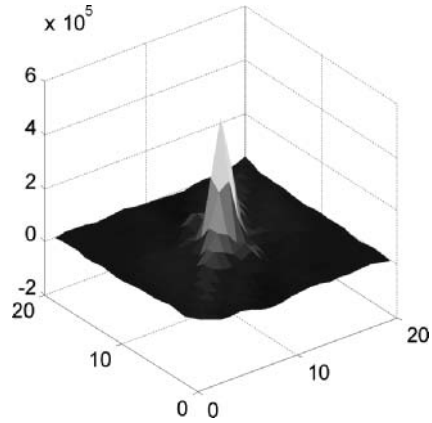


Figure 3.26 2D autocorrelation function of residual

3.4.1 Predictive image coding

Motion compensation is an example of predictive coding in which an encoder creates a prediction of a region of the current frame based on a previous or future frame and subtracts this prediction from the current region to form a residual. If the prediction is successful, the energy in the residual is lower than in the original frame and the residual can be represented with fewer bits.

Predictive coding was used as the basis for early image compression algorithms and is an important component of H.264 Intra coding, see section 3.3.2 and Chapter 6. Spatial prediction involves predicting an image sample or region from previously-transmitted samples in the same image or frame and is sometimes described as 'Differential Pulse Code Modulation' (DPCM), a term borrowed from a method of differentially encoding PCM samples in telecommunication systems.

Figure 3.27 shows a pixel X that is to be encoded. If the frame is processed in raster order, then pixels A, B and C in the current and previous rows are available in both the encoder

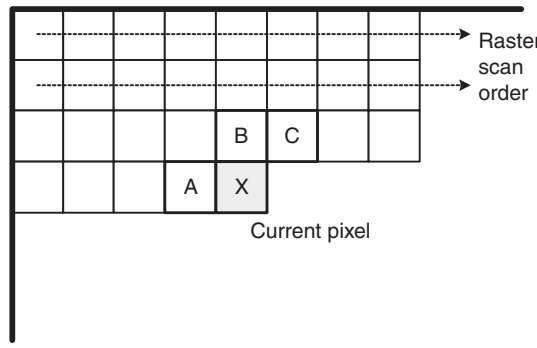


Figure 3.27 Spatial prediction (DPCM)

and the decoder since these should already have been decoded before X. The encoder forms a prediction for X based on some combination of previously coded pixels, subtracts this prediction from X and encodes the residual, the result of the subtraction. The decoder forms the same prediction and adds the decoded residual to reconstruct the pixel.

Example

Encoder prediction $P(X) = (2A + B + C)/4$

Residual $R(X) = X - P(X)$ *is encoded and transmitted.*

Decoder decodes $R(X)$ *and forms the same prediction:*

$$P(X) = (2A + B + C)/4$$

Reconstructed pixel $X = R(X) + P(X)$

If the encoding process is lossy, i.e. if the residual is quantized – see section 3.4.3, then the decoded pixels A', B' and C' may not be identical to the original A, B and C due to losses during encoding and so the above process could lead to a cumulative mismatch or 'drift' between the encoder and decoder. To avoid this, the encoder should itself decode the residual R'(X) and reconstruct each pixel. Hence the encoder uses decoded pixels A', B' and C' to form the prediction, i.e. $P(X) = (2A' + B' + C')/4$ in the above example. In this way, both encoder and decoder use the same prediction P(X) and drift is avoided.

The compression efficiency of this approach depends on the accuracy of the prediction P(X). If the prediction is accurate, i.e. P(X) is a close approximation of X, then the residual energy will be small. However, it is usually not possible to choose a predictor that works well for all areas of a complex image and better performance may be obtained by adapting the predictor depending on the local statistics of the image for example, using different predictors for areas of flat texture, strong vertical texture, strong horizontal texture, etc. It is necessary for the encoder to indicate the choice of predictor to the decoder and so there is a tradeoff between efficient prediction and the extra bits required to signal the choice of predictor.

3.4.2 Transform coding

3.4.2.1 Overview

The purpose of the transform stage in an image or video CODEC is to convert image or motion-compensated residual data into another domain, the transform domain. The choice of transform depends on a number of criteria:

1. Data in the transform domain should be decorrelated, i.e. separated into components within minimal inter-dependence, and compact, i.e. most of the energy in the transformed data should be concentrated into a small number of values.
2. The transform should be reversible.

3. The transform should be computationally tractable, e.g. low memory requirement, achievable using limited-precision arithmetic, low number of arithmetic operations, etc.

Many transforms have been proposed for image and video compression and the most popular transforms tend to fall into two categories, block-based and image-based. Examples of block-based transforms include the Karhunen-Loeve Transform (KLT), Singular Value Decomposition (SVD) and the ever-popular Discrete Cosine Transform (DCT) and its approximations [iv]. Each of these operates on blocks of N × N image or residual samples and hence the image is processed in units of a block. Block transforms have low memory requirements and are well suited to compression of block-based motion compensation residuals but tend to suffer from artefacts at block edges ('blockiness'). Image-based transforms operate on an entire image or frame or a large section of the image known as a 'tile'. The most popular image transform is the Discrete Wavelet Transform, DWT or just 'wavelet'. Image transforms such as the DWT have been shown to out-perform block transforms for still image compression but they tend to have higher memory requirements because the whole image or tile is processed as a unit and they do not necessarily 'fit' well with block-based motion compensation. The DCT and the DWT both feature in MPEG-4 Visual, with approximations to the DCT incorporated in H.264, and are discussed further in the following sections.

3.4.2.2 DCT

The Discrete Cosine Transform (DCT) operates on **X**, a block of N × N samples, typically image samples or residual values after prediction, to create **Y**, an N × N block of coefficients. The action of the DCT and its inverse, the IDCT can be described in terms of a transform matrix **A**. The forward DCT (FDCT) of an N × N sample block is given by:

$$\mathbf{Y} = \mathbf{A}\mathbf{X}\mathbf{A}^{\mathbf{T}} \tag{3.1}$$

and the inverse DCT (IDCT) by:

$$\mathbf{X} = \mathbf{A}^{\mathbf{T}}\mathbf{Y}\mathbf{A} \tag{3.2}$$

where **X** is a matrix of samples, **Y** is a matrix of coefficients and **A** is an N × N transform matrix. The elements of **A** are:

$$A_{ij} = C_i \cos\frac{(2j+1)i\pi}{2N} \quad \text{where } C_i = \sqrt{\frac{1}{N}}\ (\mathrm{i} = 0),\ C_i = \sqrt{\frac{2}{N}}\ (\mathrm{i} > 0) \tag{3.3}$$

(3.1) and (3.2) may be written in summation form:

$$Y_{xy} = C_x C_y \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} X_{ij} \cos\frac{(2j+1)y\pi}{2N} \cos\frac{(2i+1)x\pi}{2N} \quad \text{(3.4 2–D FDCT)}$$

$$X_{ij} = \sum_{x=0}^{N-1}\sum_{y=0}^{N-1} C_x C_y Y_{xy} \cos\frac{(2j+1)y\pi}{2N} \cos\frac{(2i+1)x\pi}{2N} \quad \text{(3.5 2–D IDCT)}$$

Example: N = 4

The transform matrix **A** for a 4 × 4 DCT is:

$$A = \begin{bmatrix} \frac{1}{2}\cos(0) & \frac{1}{2}\cos(0) & \frac{1}{2}\cos(0) & \frac{1}{2}\cos(0) \\ \sqrt{\frac{1}{2}}\cos\left(\frac{\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{3\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{5\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{7\pi}{8}\right) \\ \sqrt{\frac{1}{2}}\cos\left(\frac{2\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{6\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{10\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{14\pi}{8}\right) \\ \sqrt{\frac{1}{2}}\cos\left(\frac{3\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{9\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{15\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{21\pi}{8}\right) \end{bmatrix} \tag{3.6}$$

The cosine function is symmetrical and repeats after 2π radians and hence **A** can be simplified to:

$$A = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \sqrt{\frac{1}{2}}\cos\left(\frac{\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{3\pi}{8}\right) & -\sqrt{\frac{1}{2}}\cos\left(\frac{3\pi}{8}\right) & -\sqrt{\frac{1}{2}}\cos\left(\frac{\pi}{8}\right) \\ \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ \sqrt{\frac{1}{2}}\cos\left(\frac{3\pi}{8}\right) & -\sqrt{\frac{1}{2}}\cos\left(\frac{\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{\pi}{8}\right) & -\sqrt{\frac{1}{2}}\cos\left(\frac{3\pi}{8}\right) \end{bmatrix} \tag{3.7}$$

or

$$A = \begin{bmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & -c \end{bmatrix} \quad \text{where} \quad \begin{aligned} a &= \frac{1}{2} \\ b &= \sqrt{\frac{1}{2}}\cos\left(\frac{\pi}{8}\right) \\ c &= \sqrt{\frac{1}{2}}\cos\left(\frac{3\pi}{8}\right) \end{aligned} \tag{3.8}$$

Evaluating the cosines gives:

$$A = \begin{bmatrix} 0.5 & 0.5 & 0.5 & 0.5 \\ 0.653 & 0.271 & -0.271 & -0.653 \\ 0.5 & -0.5 & -0.5 & 0.5 \\ 0.271 & -0.653 & 0.653 & -0.271 \end{bmatrix}$$

The output of a 2-dimensional FDCT is a set of N × N coefficients representing the image block data in the DCT domain which can be considered as 'weights' of a set of standard **basis patterns**. The basis patterns for the 4 × 4 and 8 × 8 DCTs are shown in Figure 3.28 and Figure 3.29 respectively and are composed of combinations of horizontal and vertical cosine functions. Any image block may be reconstructed by combining all N × N basis patterns, with each basis multiplied by the appropriate weighting factor (coefficient).

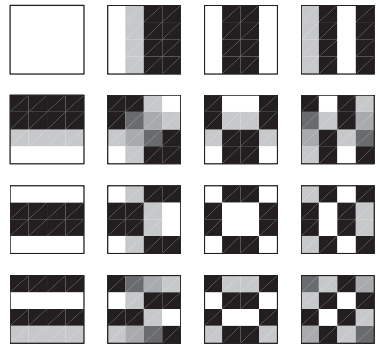


Figure 3.28 4 × 4 DCT basis patterns

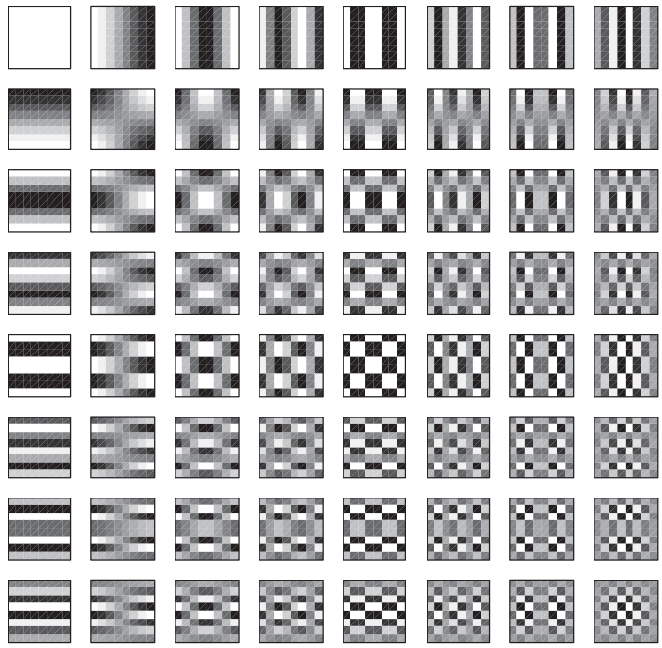


Figure 3.29 8 × 8 DCT basis patterns

Example 1 Calculating the DCT of a 4 × 4 block

Let **X** be a 4 × 4 block of samples from an image:

i= \ J =	0	1	2	3
0	5	11	8	10
1	9	8	4	12
2	1	10	11	4
3	19	6	15	7

The Forward DCT of **X** is given by: $\mathbf{Y} = \mathbf{AXA^T}$. The first matrix multiplication, $\mathbf{Y}' = \mathbf{AX}$, corresponds to calculating the 1-dimensional DCT of each **column** of **X**. For example, Y'_{00} is calculated as follows:

$$Y'_{00} = A_{00}X_{00} + A_{01}X_{10} + A_{02}X_{20} + A_{03}X_{30} =$$
$$(0.5 * 5) + (0.5 * 9) + (0.5 * 1) + (0.5 * 19) = 17.0$$

The complete result of the column calculations is:

$$Y' = AX = \begin{bmatrix} 17 & 17.5 & 19 & 16.5 \\ -6.981 & 2.725 & -6.467 & 4.125 \\ 7 & -0.5 & 4 & 0.5 \\ -9.015 & 2.660 & 2.679 & -4.414 \end{bmatrix}$$

Carrying out the second matrix multiplication, $\mathbf{Y} = \mathbf{Y}'\mathbf{A^T}$, is equivalent to carrying out a 1-D DCT on each **row** of Y':

$$Y = AXA^T = \begin{bmatrix} 35.0 & -0.079 & -1.5 & 1.115 \\ -3.299 & -4.768 & 0.443 & -9.010 \\ 5.5 & 3.029 & 2.0 & 4.699 \\ -4.045 & -3.010 & -9.384 & -1.232 \end{bmatrix}$$

Note that the order of the row and column calculations does not affect the final result.

Example 2 Image block and DCT coefficients

Figure 3.30 shows an image with a 4 × 4 block selected and Figure 3.31 shows the block in close-up, together with the DCT coefficients. The advantage of representing the block in the DCT domain is not immediately obvious since there is no reduction in the amount of data;

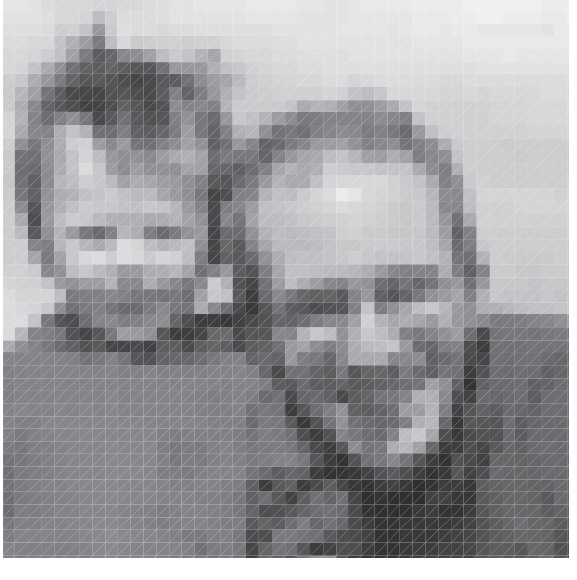


Figure 3.30 Image section showing 4 × 4 block

126	159	178	181
98	151	181	181
80	137	176	156
75	114	88	68

Original block

537.2	-76.0	-54.8	-7.8
-106.1	35.0	-12.7	-6.1
-42.7	46.5	10.3	-9.8
-20.2	12.9	3.9	-8.5

DCT coefficients

Figure 3.31 Close-up of 4 × 4 block; DCT coefficients

134	134	134	134
134	134	134	134
134	134	134	134
134	134	134	134

1 coefficient

169	169	169	169
149	149	149	149
120	120	120	120
100	100	100	100

2 coefficients

144	159	179	194
124	138	159	173
95	110	130	145
75	89	110	124

3 coefficients

146	179	187	165
95	146	179	175
66	117	150	146
76	109	117	96

5 coefficients

Figure 3.32 Block reconstructed from (a) 1, (b) 2, (c) 3, (d) 5 coefficients

instead of 16 pixel values, we need to store 16 DCT coefficients. The usefulness of the DCT becomes clear when the block is reconstructed from a subset of the coefficients.

Setting all the coefficients to zero except the most significant, coefficient (0,0) described as the 'DC' coefficient, and performing the IDCT gives the output block shown in Figure 3.32 (a), the mean of the original pixel values. Calculating the IDCT of the two most significant coefficients gives the block shown in Figure 3.32 (b). Adding more coefficients before calculating the IDCT produces a progressively more accurate reconstruction of the original block and by the time five coefficients are included (Figure 3.32 (d)), the reconstructed block is a reasonably close match to the original. Hence it is possible to reconstruct an approximate copy of the block from a subset of the 16 DCT coefficients. Removing the coefficients with insignificant magnitudes, for example by quantization, see section 3.4.3, enables image data to be represented with a reduced number of coefficient values at the expense of some loss of quality.

3.4.2.3 Wavelet

The 'wavelet transform' that is popular in image compression is based on sets of filters with coefficients that are equivalent to discrete wavelet functions [v]. The basic operation of the transform is as follows, applied to a discrete signal containing N samples. A pair of filters is applied to the signal to decompose it into a low frequency band (L) and a high frequency band (H). Each band is subsampled by a factor of two, so that the two frequency bands each contain N/2 samples. With the correct choice of filters, this operation is reversible.

This approach may be extended to apply to a 2-dimensional signal such as an intensity image (Figure 3.33). Each row of a 2D image is filtered with a low-pass and a high-pass filter (L_x and H_x) and the output of each filter is down-sampled by a factor of two to produce the intermediate images L and H. L is the original image low-pass filtered and downsampled in the x-direction and H is the original image high-pass filtered and downsampled in the x-direction. Next, each column of these new images is filtered with low- and high-pass filters

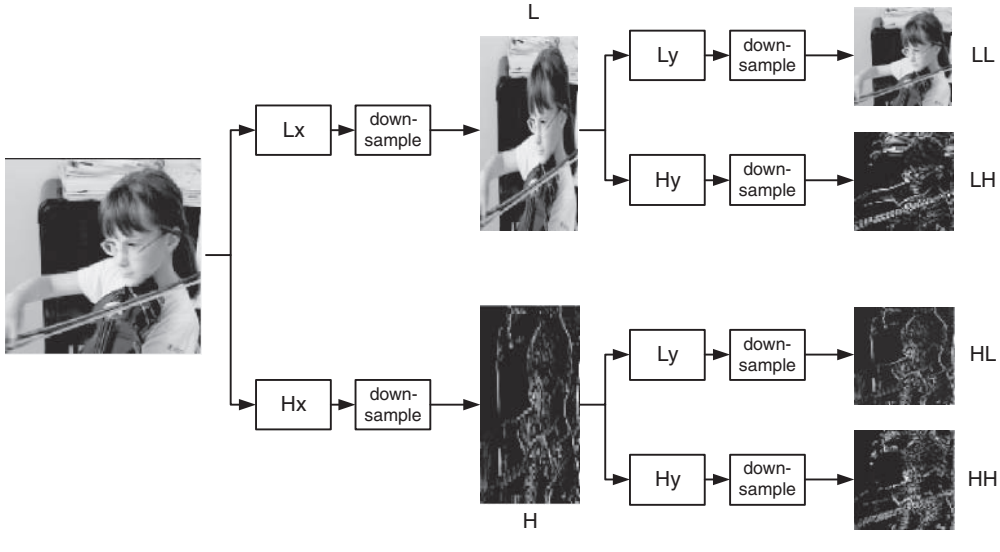


Figure 3.33 Two-dimensional wavelet decomposition process

Figure 3.34 Image after one level of decomposition

L_y and H_y and down-sampled by a factor of two to produce four sub-images LL, LH, HL and HH. These four 'sub-band' images can be combined to create an output image with the same number of samples as the original (Figure 3.34). 'LL' is the original image, low-pass filtered in horizontal and vertical directions and subsampled by a factor of two. 'HL' is high-pass filtered in the vertical direction and contains residual vertical frequencies, 'LH' is high-pass filtered in the horizontal direction and contains residual horizontal frequencies and 'HH' is high-pass filtered in both horizontal and vertical directions. Between them, the four sub-band images contain all of the information present in the original image but the sparse nature of the LH, HL and HH sub-bands makes them amenable to compression.

In an image compression application, the 2-dimensional wavelet decomposition described above is applied again to the 'LL' image, forming four new sub-band images. The resulting low-pass image, always the top-left sub-band image, is iteratively filtered to create a tree of sub-band images. Figure 3.35 shows the result of two stages of this decomposition and

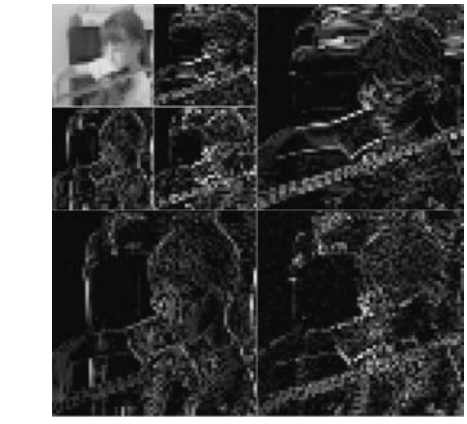


Figure 3.35 Two-stage wavelet decomposition of image

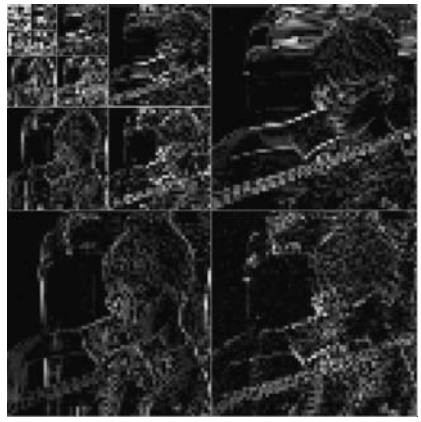


Figure 3.36 Five-stage wavelet decomposition of image

Figure 3.36 shows the result of five stages of decomposition. Many of the samples (coefficients) in the higher-frequency sub-band images are close to zero, shown here as near-black, and it is possible to achieve compression by removing these insignificant coefficients prior to transmission. At the decoder, the original image is reconstructed by repeated up-sampling, filtering and addition, reversing the order of operations shown in Figure 3.33.

3.4.3 Quantization

A quantizer maps a signal with a range of values X to a quantized signal with a reduced range of values Y. It should be possible to represent the quantized signal with fewer bits than the original since the range of possible values is smaller. A **scalar quantizer** maps one sample of the input signal to one quantized output value and a **vector quantizer** maps a group of input samples, a 'vector', to a group of quantized values.

3.4.3.1 Scalar quantization

A simple example of scalar quantization is the process of rounding a fractional number to the nearest integer, i.e. the mapping is from R to Z. The process is lossy (not reversible) since it is not possible to determine the exact value of the original fractional number from the rounded integer.

A more general example of a uniform quantizer is:

$$\begin{aligned} FQ &= round\left(\frac{X}{QP}\right) \\ Y &= FQ.QP \end{aligned} \tag{3.9}$$

where QP is a quantization 'step size'. The quantized output levels are spaced at uniform intervals of QP as shown in the following example.

Example Y = QP.round(X/QP)

	Y			
X	QP=1	QP=2	QP=3	QP=5
−4	−4	−4	−3	−5
−3	−3	−2	−3	−5
−2	−2	−2	−3	0
−1	−1	0	0	0
0	0	0	0	0
1	1	0	0	0
2	2	2	3	0
3	3	2	3	5
4	4	4	3	5
5	5	4	6	5
6	6	6	6	5
7	7	6	6	5
8	8	8	9	10
9	9	8	9	10
10	10	10	9	10
11	11	10	12	10
.....				

Figure 3.37 shows two examples of scalar quantizers, a linear quantizer with a uniform mapping between input and output values and a non-linear quantizer that has a 'dead zone' about zero, in which small-valued inputs are mapped to zero.

In image and video compression CODECs, the quantization operation is usually made up of two parts, a forward quantizer FQ in the encoder and an 'inverse quantizer' or 'rescaler' (IQ) in the decoder. A critical parameter is the **step size** QP between successive re-scaled values. If the step size is large, the range of quantized values is small and can therefore be efficiently represented and hence highly compressed during transmission, but the re-scaled values are a crude approximation to the original signal. If the step size is small, the re-scaled values match the original signal more closely but the larger range of quantized values reduces compression efficiency.

Quantization may be used to reduce the precision of image data after applying a transform such as the DCT or wavelet transform and to remove insignificant values such as near-zero DCT or wavelet coefficients. The forward quantizer in an image or video encoder is designed to map insignificant coefficient values to zero whilst retaining a small number of significant, non-zero coefficients. The output of a forward quantizer is therefore typically a 'sparse' array of quantized coefficients, mainly containing zeros.

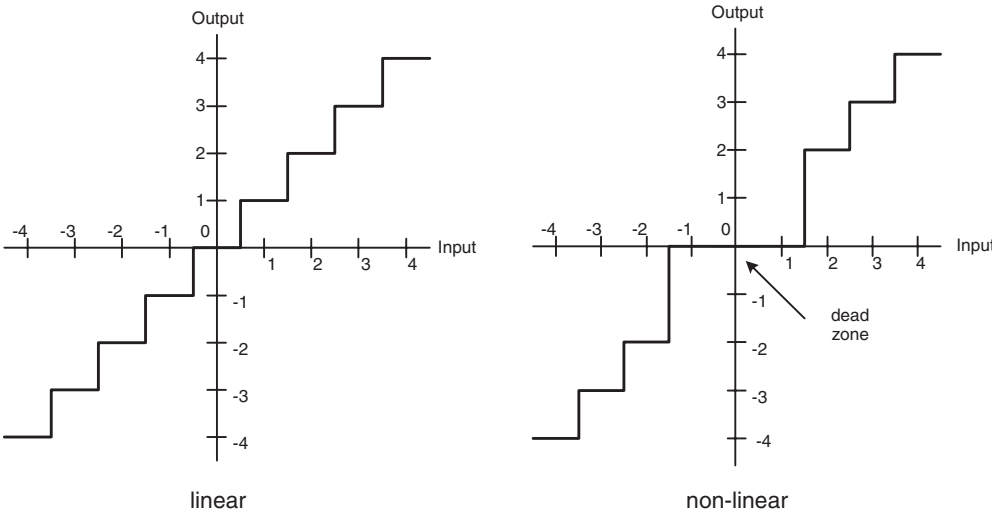


Figure 3.37 Scalar quantizers: linear; non-linear with dead zone

3.4.3.2 Vector quantization

A vector quantizer maps a set of input data such as a block of image samples to a single value (codeword) and at the decoder, each codeword maps to an approximation to the original set of input data, a 'vector'. The set of vectors are stored at the encoder and decoder in a codebook. A typical application of vector quantization to image compression [vi] is as follows:

1. Partition the original image into regions such as N × N pixel blocks.
2. Choose a vector from the codebook that matches the current region as closely as possible.
3. Transmit an index that identifies the chosen vector to the decoder.
4. At the decoder, reconstruct an approximate copy of the region using the selected vector.

A basic system is illustrated in Figure 3.38. Here, quantization is applied in the image (spatial) domain, i.e. groups of image samples are quantized as vectors, but it can equally be applied to motion compensated and/or transformed data. Key issues in vector quantizer design include the design of the codebook and efficient searching of the codebook to find the optimal vector.

3.4.4 Reordering and zero encoding

Quantized transform coefficients are required to be encoded as compactly as possible prior to storage and transmission. In a transform-based image or video encoder, the output of the quantizer is a sparse array containing a few non-zero coefficients and a large number of zero-valued coefficients. Re-ordering to group together non-zero coefficients and efficient encoding of zero coefficients are applied prior to entropy encoding.

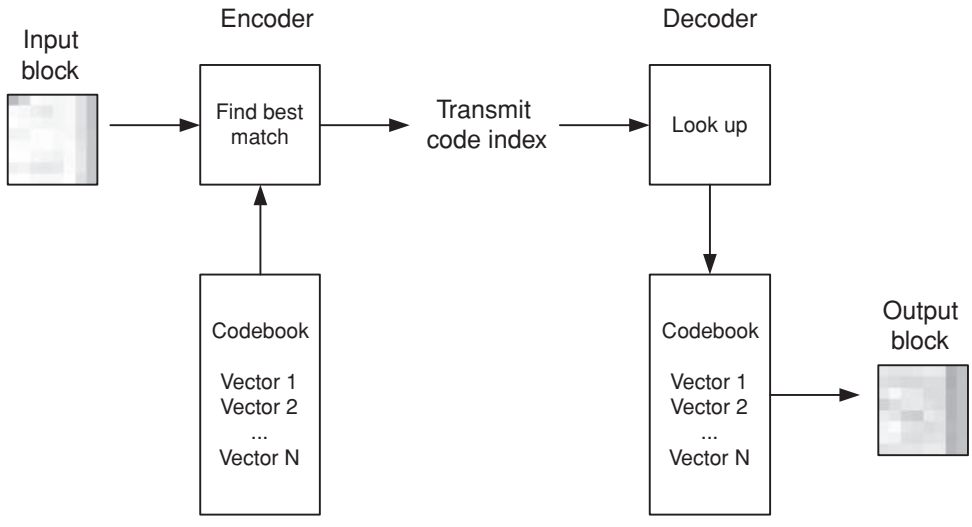


Figure 3.38 Vector quantization

3.4.4.1 DCT

Coefficient distribution

The significant DCT coefficients of a block of image or residual samples are typically the 'low frequency' positions around the DC (0,0) coefficient.

Figure 3.39 plots the probability of non-zero DCT coefficients at each position in an 8 × 8 block in a QCIF residual frame (Figure 3.6). The non-zero DCT coefficients are clustered around the top-left (DC) coefficient and the distribution is roughly symmetrical in the horizontal and vertical directions.

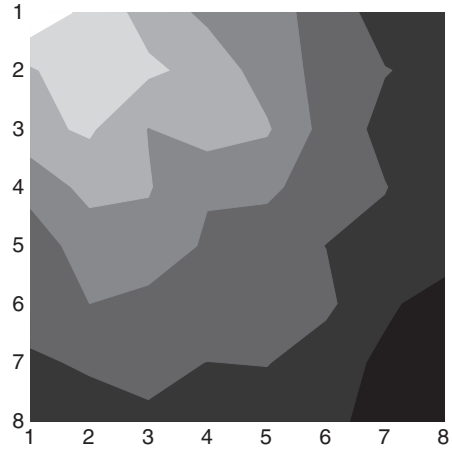


Figure 3.39 8 × 8 DCT coefficient distribution (frame)

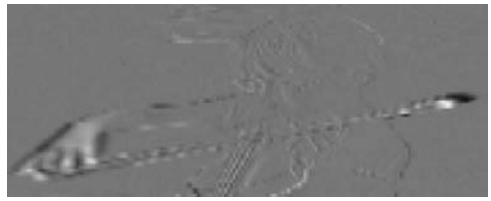


Figure 3.40 Residual field picture

Figure 3.41 plots the probability of non-zero DCT coefficients for a residual field (Figure 3.40); here, the coefficients are clustered around the DC position but are 'skewed', i.e. more non-zero coefficients occur along the left-hand edge of the plot. This is because a field picture may have a stronger high-frequency component in the vertical axis due to the subsampling in the vertical direction, resulting in larger DCT coefficients corresponding to vertical frequencies (Figure 3.28).

Scan

After quantization, the DCT coefficients for a block are reordered to group together non-zero coefficients, enabling efficient representation of the remaining zero-valued quantized coefficients. The optimum re-ordering path or scan order depends on the distribution of non-zero DCT coefficients. For a typical frame block with a distribution similar to Figure 3.39, a suitable scan order is a zigzag starting from the DC or top-left coefficient. Starting with the DC coefficient, each quantized coefficient is copied into a one-dimensional array in the order shown in Figure 3.42. Non-zero coefficients tend to be grouped together at the start of the re-ordered array, followed by long sequences of zeros.

The zig-zag scan may not be ideal for a field block because of the skewed coefficient distribution, Figure 3.41, and a modified scan order such as Figure 3.43 may be more effective for some field blocks, in which coefficients on the left hand side of the block are scanned before the right hand side.

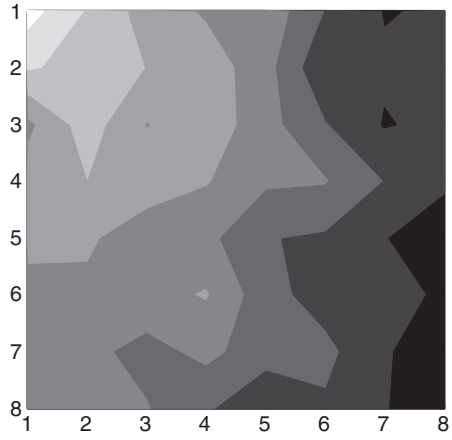


Figure 3.41 8 × 8 DCT coefficient distribution (field)

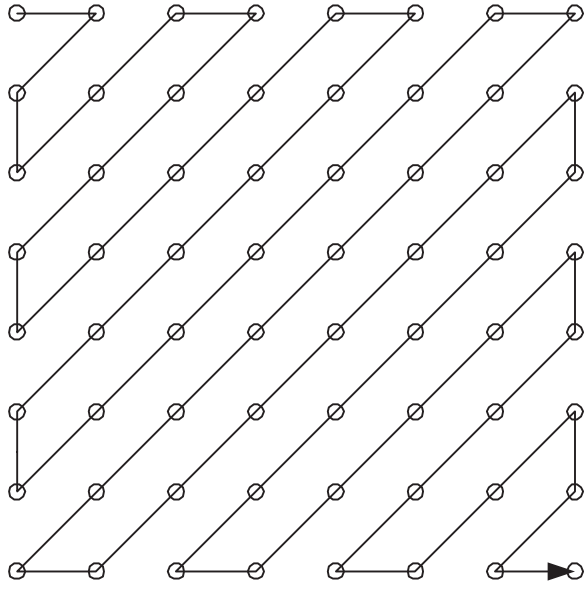


Figure 3.42 Zigzag scan example : frame block

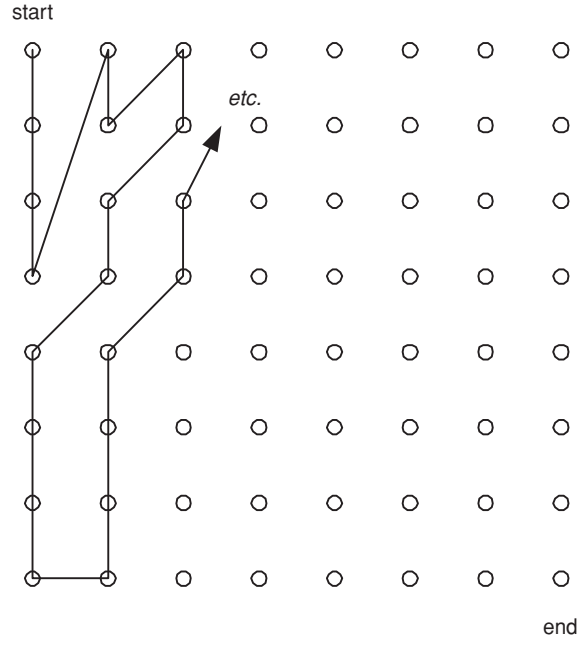


Figure 3.43 Zigzag scan example : field block

Run-Level Encoding

The output of the re-ordering process is an array that typically contains one or more clusters of non-zero coefficients near the start, followed by strings of zero coefficients. The large number of zero values may be encoded to represent them more compactly. The array of re-ordered coefficients are represented as (run,level) pairs where **run** indicates the number of zeros preceding a non-zero coefficient and **level** indicates the magnitude of the non-zero coefficient.

Example

1. Input array: 16,0,0,−3,5,6,0,0,0,0,−7,. . .
2. Output values: (0,16),(2,−3),(0,5),(0,6),(4,−7). . .
3. Each of these output values (a run-level pair) is encoded as a separate symbol by the entropy encoder.

Higher-frequency DCT coefficients are very often quantized to zero and so a reordered block will usually end in a run of zeros. A special case is required to indicate the final non-zero coefficient in a block. If 'two-dimensional' run-level encoding is used, each run-level pair is encoded as above and a separate code symbol, '**last**', indicates the end of the non-zero values. If 'three-dimensional' run-level encoding is used, each symbol encodes three quantities, **run**, **level** and **last**. In the example above, if –7 is the final non-zero coefficient, the 3-D values are:

$$(0,\ 16,\ 0),\ (2,\ -3,\ 0), (0,\ 5,\ 0),\ (0,\ 6,\ 0),\ (4,\ -7,\ 1)$$

The 1 in the final code indicates that this is the last non-zero coefficient in the block.

3.4.4.2 Wavelet

Coefficient distribution

Figure 3.36 shows a typical distribution of 2D wavelet coefficients. Many coefficients in higher sub-bands, towards the bottom-right of the figure, are near zero and may be quantized to zero without significant loss of image quality. Non-zero coefficients tend to be related to structures in the image; for example, the violin bow appears as a clear horizontal structure in all the horizontal and diagonal sub-bands. When a coefficient in a lower-frequency sub-band is non-zero, there is a strong probability that coefficients in the corresponding position in higher-frequency sub-bands will also be non-zero. We may consider a 'tree' of non-zero quantized coefficients, starting with a 'root' in a low-frequency sub-band. Figure 3.44 illustrates this concept. A single coefficient in the LL band of layer 1 has one corresponding coefficient in each of the other bands of layer 1, i.e. these four coefficients correspond to the same region in the original image. The layer 1 coefficient position maps to four corresponding child coefficient positions in each sub-band at layer 2. Recall that the layer 2 sub-bands have twice the horizontal and vertical resolution of the layer 1 sub-bands.

Zerotree encoding

It is desirable to encode the non-zero wavelet coefficients as compactly as possible prior to entropy coding [vii]. An efficient way of achieving this is to encode each tree of non-zero coefficients starting from the lowest or root level of the decomposition. A coefficient at the

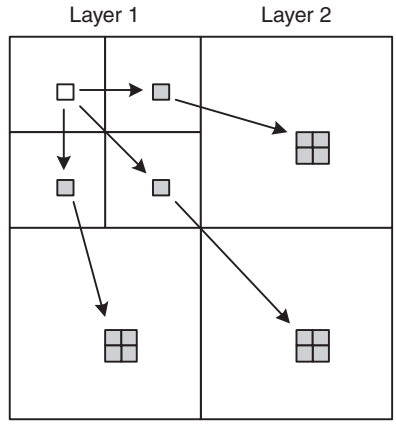


Figure 3.44 Wavelet coefficient and 'children'

lowest layer is encoded, followed by its child coefficients at the next layer up, and so on. The encoding process continues until the tree reaches a zero-valued coefficient. Further children of a zero-valued coefficient are likely to be zero themselves and so the remaining children are represented by a single code that identifies a tree of zeros (**zerotree**). The decoder reconstructs the coefficient map starting from the root of each tree; non-zero coefficients are decoded and reconstructed and when a zerotree code is reached, all remaining 'children' are set to zero. This is the basis of the **embedded zero tree** (EZW) method of encoding wavelet coefficients. An extra possibility is included in the encoding process, where a zero coefficient may be followed by (a) a zero tree, as before or (b) a non-zero child coefficient. Case (b) does not occur very often but reconstructed image quality is slightly improved by catering for the occasional occurrences of case (b).

3.5 Entropy coder

The entropy encoder converts a series of symbols representing elements of the video sequence into a compressed bitstream suitable for transmission or storage. Input symbols may include quantized transform coefficients, run-level or zerotree encoded as described in section 3.4.4, motion vectors with integer or sub-pixel resolution, marker codes that indicate a resynchronization point in the sequence, macroblock headers, picture headers, sequence headers etc and supplementary information, 'side' information that is not essential for correct decoding.

3.5.1 Predictive coding

Certain symbols are highly correlated in local regions of the picture. For example, the average or DC value of neighbouring intra-coded blocks of pixels may be very similar, neighbouring motion vectors may have similar x and y displacements and so on. Coding efficiency can be improved by predicting elements of the current block or macroblock from previously-encoded data and encoding the difference between the prediction and the actual value.

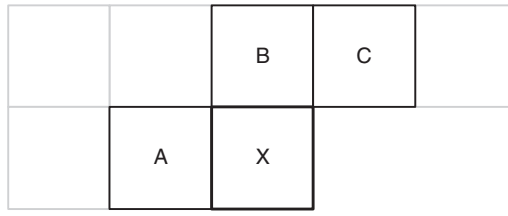


Figure 3.45 Motion vector prediction candidates

The motion vector for a block or macroblock indicates the offset to a prediction reference in a previously encoded frame. Vectors for neighbouring blocks or macroblocks are often correlated because object motion may extend across large regions of a frame. This is especially true for small block sizes, e.g. 4 × 4 block vectors (Figure 3.22) and/or for large moving objects. Compression of the motion vector field may be improved by predicting each motion vector from previously encoded vectors. A simple prediction for the vector of the current macroblock X is the horizontally adjacent macroblock A (Figure 3.45); alternatively three or more previously-coded vectors may be used to predict the vector at macroblock X, e.g. A, B and C in Figure 3.45. The difference between the predicted and actual motion vector, the Motion Vector Difference or MVD, is encoded and transmitted.

The quantization parameter or quantizer step size controls the trade-off between compression efficiency and image quality. In a real-time video CODEC it may be necessary to modify the quantization within an encoded frame, for example to change the compression ratio in order to match the coded bit rate to a transmission channel rate. It is usually sufficient to change the parameter by a small amount between successive coded macroblocks. The modified quantization parameter must be signalled to the decoder and instead of sending a new quantization parameter value, it may be preferable to send a delta or difference value, e.g. +/−1 or +/−2, indicating the change required. Fewer bits are required to encode a small delta value than to encode a completely new quantization parameter.

3.5.2 Variable-length coding

A variable-length encoder maps input symbols to a series of codewords, variable length codes or VLCs. Each symbol maps to a codeword and codewords may have varying length but must each contain an integral number of bits. Frequently-occurring symbols are represented with short VLCs whilst less common symbols are represented with long VLCs. Over a sufficiently large number of encoded symbols this leads to compression of the data.

3.5.2.1 Huffman coding

Huffman coding assigns a VLC to each symbol based on the probability of occurrence of different symbols. According to the original scheme proposed by Huffman in 1952 [viii], it is necessary to calculate the probability of occurrence of each symbol and to construct a set of variable length codewords. This process will be illustrated by two examples.

Table 3.2 Probability of occurrence of motion vectors in sequence 1

Vector	Probability p	$\log_2(1/p)$
−2	0.1	3.32
−1	0.2	2.32
0	0.4	1.32
1	0.2	2.32
2	0.1	3.32

Example 1: Huffman coding, Sequence 1 motion vectors

The motion vector difference data (MVD) for video sequence 1 is required to be encoded. Table 3.2 lists the probabilities of the most commonly-occurring motion vectors in the encoded sequence and their **information content**, $\log_2(1/p)$. To achieve optimum compression, each value should be represented with exactly $\log_2(1/p)$ bits. '0' is the most common value and the probability drops for larger motion vectors. This distribution is representative of a sequence containing moderate motion.

1. Generating the Huffman code tree

To generate a Huffman code table for this set of data, the following iterative procedure is carried out:

1. Order the list of data in increasing order of probability.
2. Combine the two lowest-probability data items into a 'node' and assign the joint probability of the data items to this node.
3. Re-order the remaining data items and node(s) in increasing order of probability and repeat step 2.

The procedure is repeated until there is a single 'root' node that contains all other nodes and data items listed 'beneath' it. This procedure is illustrated in Figure 3.46.

Original list:	The data items are shown as square boxes. Vectors (−2) and (+2) have the lowest probability and these are the first candidates for merging to form node 'A'.
Stage 1:	The newly-created node 'A', shown as a circle, has a probability of 0.2, from the combined probabilities of (−2) and (2). There are now three items with probability 0.2. Choose vectors (−1) and (1) and merge to form node 'B'.
Stage 2:	A now has the lowest probability (0.2) followed by B and the vector 0; choose A and B as the next candidates for merging to form 'C'.
Stage 3:	Node C and vector (0) are merged to form 'D'.
Final tree:	The data items have all been incorporated into a binary 'tree' containing five data values and four nodes. Each data item is a 'leaf' of the tree.

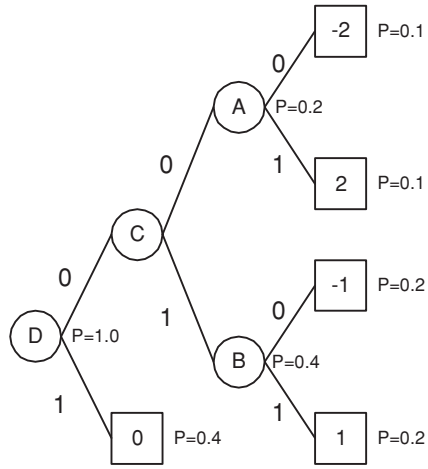


Figure 3.46 Generating the Huffman code tree: Sequence 1 motion vectors

2. Encoding

Each 'leaf' of the binary tree is mapped to a variable-length code. To find this code, the tree is traversed from the root node, D in this case, to the leaf or data item. For every branch, a 0 or 1 is appended to the code, 0 for an upper branch, 1 for a lower branch, shown in the final tree of Figure 3.46, giving the following set of codes (Table 3.3).

Encoding is achieved by transmitting the appropriate code for each data item. Note that once the tree has been generated, the codes may be stored in a look-up table.

High probability data items are assigned short codes, e.g. 1 bit for the most common vector '0'. However, the vectors (−2, 2, −1, 1) are each assigned 3-bit codes despite the fact that –1 and 1 have higher probabilities than –2 and 2. The lengths of the Huffman codes, each an integral number of bits, do not match the ideal lengths given by $\log_2(1/p)$. No code contains any other code as a prefix, which means that, reading from the left-hand bit, each code is uniquely decodable.

For example, the series of vectors (1, 0, −2) would be transmitted as the binary sequence 0111000.

Table 3.3 Huffman codes for sequence 1 motion vectors

Vector	Code	Bits (actual)	Bits (ideal)
0	1	1	1.32
1	011	3	2.32
−1	010	3	2.32
2	001	3	3.32
−2	000	3	3.32

Table 3.4 Probability of occurrence of motion vectors in sequence 2

Vector	Probability	$\log_2(1/p)$
−2	0.02	5.64
−1	0.07	3.84
0	0.8	0.32
1	0.08	3.64
2	0.03	5.06

3. Decoding

In order to decode the data, the decoder must have a local copy of the Huffman code tree or look-up table. This may be achieved by transmitting the look-up table itself or by sending the list of data and probabilities prior to sending the coded data. Each uniquely-decodeable code is converted back to the original data, for example:

1. 011 is decoded as (1)
2. 1 is decoded as (0)
3. 000 is decoded as (−2).

Example 2: Huffman coding, sequence 2 motion vectors

Repeating the process described above for a second sequence with a different distribution of motion vector probabilities gives a different result. The probabilities are listed in Table 3.4 and note that the zero vector is much more likely to occur in this example, representative of a sequence with little movement.

The corresponding Huffman tree is given in Figure 3.47. The 'shape' of the tree has changed because of the distribution of probabilities and this gives a different set of Huffman codes, shown in Table 3.5. There are still four nodes in the tree, one less than the number of data items (5), as is always the case with Huffman coding.

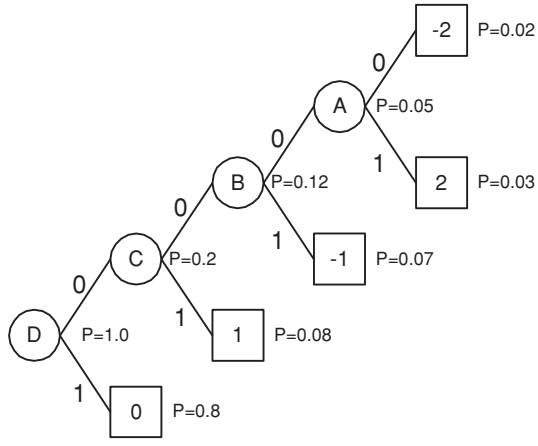


Figure 3.47 Huffman tree for sequence 2 motion vectors

Table 3.5 Huffman codes for sequence 2 motion vectors

Vector	Code	Bits (actual)	Bits (ideal)
0	1	1	0.32
1	01	2	3.64
−1	001	3	3.84
2	0001	4	5.06
−2	0000	4	5.64

If the probability distributions are accurate, Huffman coding provides a relatively compact representation of the original data. In these examples, the frequently occurring (0) vector is represented efficiently as a single bit. However, to achieve optimum compression, a separate code table is required for each of the two sequences because of their different probability distributions. The loss of potential compression efficiency due to the requirement for integral-length codes is very clear for vector '0' in sequence 2, since the optimum number of bits (information content) is 0.32 but the best that can be achieved with Huffman coding is 1 bit.

3.5.2.2 Pre-calculated Huffman-based coding

The Huffman coding process has two disadvantages for a practical video CODEC. First, the decoder must use the same codeword set as the encoder. This means that the encoder needs to transmit the information contained in the probability table before the decoder can decode the bit stream and this extra overhead reduces compression efficiency, particularly for shorter video sequences. Second, the probability table for a large video sequence, required to generate the Huffman tree, cannot be calculated until after the video data is encoded which may introduce an unacceptable delay into the encoding process. For these reasons, image and video coding standards define sets of codewords based on the probability distributions of 'generic' video material. The following two examples of pre-calculated VLC tables are taken from MPEG-4 Visual (Simple Profile).

Transform Coefficients (TCOEF)

MPEG-4 Visual uses 3-D coding of quantized coefficients in which each codeword represents a combination of (run, level, last). A total of 102 specific combinations of (run, level, last) have VLCs assigned to them and 26 of these codes are shown in Table 3.6.

A further 76 VLCs are defined, each up to 13 bits long. The last bit of each codeword is the sign bit 's', indicating the sign of the decoded coefficient, where 0=positive and 1=negative. Any (run, level, last) combination that is not listed in the table is coded using an escape sequence, a special ESCAPE code (0000011) followed by a 13-bit fixed length code describing the values of run, level and last.

Some of the codes shown in Table 3.6 are represented in 'tree' form in Figure 3.48. A codeword containing a run of more than eight zeros is not valid, hence any codeword starting with 000000000... indicates an error in the bitstream or possibly a start code, which begins with a long sequence of zeros, occurring at an unexpected position in the sequence. All other sequences of bits can be decoded as valid codes. Note that the smallest codes are allocated to

Table 3.6 MPEG-4 Visual Transform Coefficient (TCOEF) VLCs : partial, all codes < 9 bits

Last	Run	Level	Code
0	0	1	10s
0	1	1	110s
0	2	1	1110s
0	0	2	1111s
1	0	1	0111s
0	3	1	01101s
0	4	1	01100s
0	5	1	01011s
0	0	3	010101s
0	1	2	010100s
0	6	1	010011s
0	7	1	010010s
0	8	1	010001s
0	9	1	010000s
1	1	1	001111s
1	2	1	001110s
1	3	1	001101s
1	4	1	001100s
0	0	4	0010111s
0	10	1	0010110s
0	11	1	0010101s
0	12	1	0010100s
1	5	1	0010011s
1	6	1	0010010s
1	7	1	0010001s
1	8	1	0010000s
ESCAPE			0000011s
...	...	...	...

short runs and small levels since these occur most frequently, e.g. code '10' represents a run of 0 and a level of +/−1.

Motion Vector Difference (MVD)

Differentially coded motion vectors (MVD) are each encoded as a pair of VLCs, one for the x-component and one for the y-component. Part of the table of VLCs is shown in Table 3.7. A further 49 codes, 8–13 bits long, are not shown here. Note that the shortest codes represent small motion vector differences, e.g. MVD=0 is represented by a single bit code '1'.

These code tables are clearly similar to 'true' Huffman codes since each symbol is assigned a unique codeword, common symbols are assigned shorter codewords and, within a table, no codeword is the prefix of any other codeword. The main differences from 'true' Huffman coding are (a) the codewords are pre-calculated based on 'generic' probability distributions and (b) in the case of TCOEF, only 102 commonly-occurring symbols have defined codewords with any other symbol encoded using a fixed-length code.

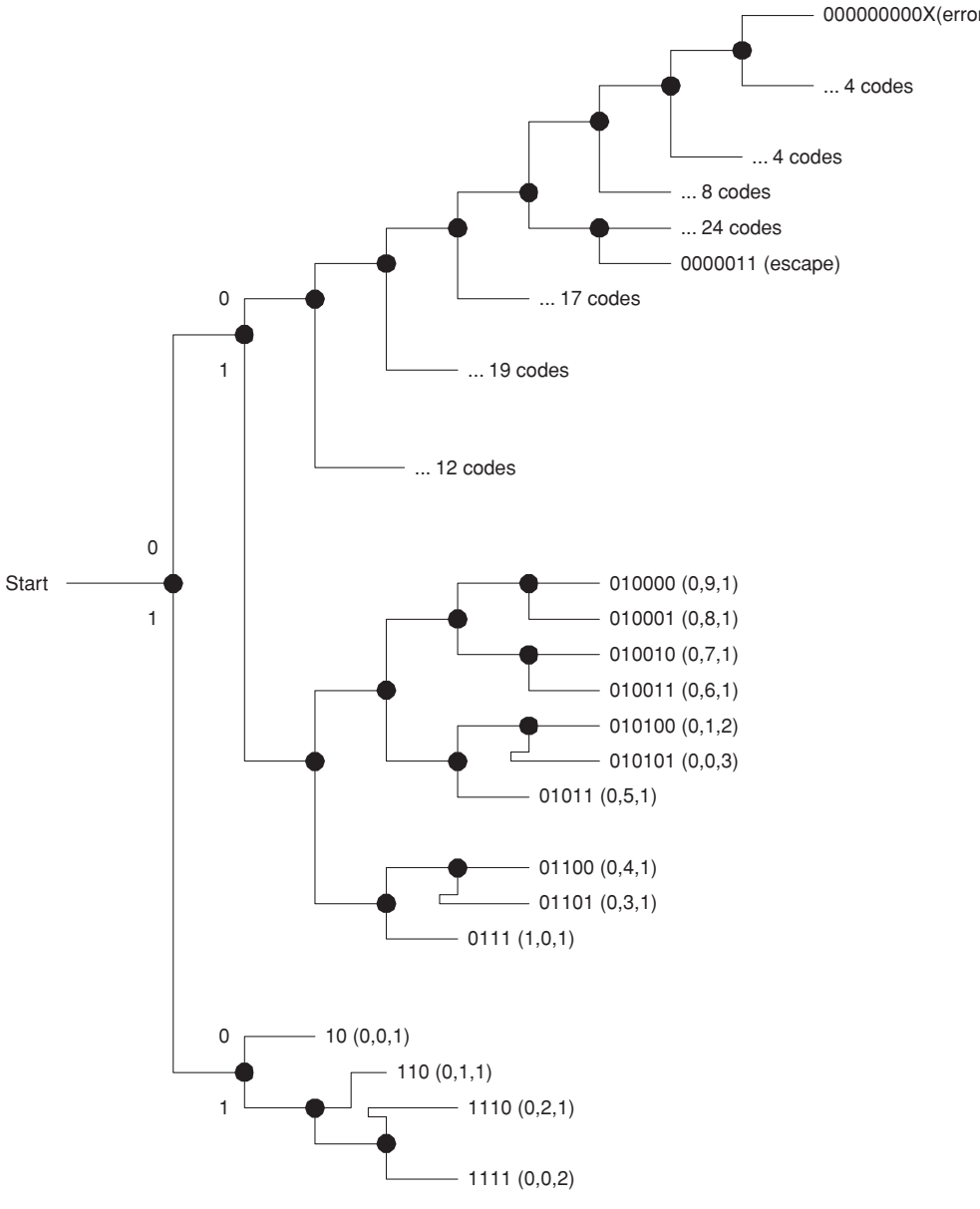


Figure 3.48 MPEG4 TCOEF VLCs (partial)

3.5.2.3 Other variable-length codes

As well as Huffman and Huffman-based codes, a number of other families of VLCs are of interest in video coding applications. One serious disadvantage of Huffman-based codes for transmission of coded data is that they are sensitive to transmission errors. An error in a sequence of VLCs may cause a decoder to lose synchronization and fail to decode subsequent codes correctly, leading to spreading or propagation of an error in a decoded

Table 3.7 MPEG4 Motion Vector Difference (MVD) VLCs

MVD	Code
0	1
+0.5	010
−0.5	011
+1	0010
−1	0011
+1.5	00010
−1.5	00011
+2	0000110
−2	0000111
+2.5	00001010
−2.5	00001011
+3	00001000
−3	00001001
+3.5	00000110
−3.5	00000111
...	...

sequence. Reversible VLCs (RVLCs) that can be successfully decoded in either a forward or a backward direction can dramatically improve decoding performance when errors occur. A drawback of pre-defined code tables such as Table 3.6 and Table 3.7 is that both encoder and decoder must store the table in some form. An alternative approach is to use codes that can be generated automatically 'on the fly' if the input symbol is known. Exponential Golomb codes (Exp-Golomb) fall into this category and are described in Chapter 7.

3.5.3 Arithmetic coding

The variable length coding schemes described in section 3.5.2 share the fundamental disadvantage that assigning a codeword containing an integral number of bits to each symbol is sub-optimal, since the optimal number of bits for a symbol depends on the information content and is usually a fractional number. Compression efficiency of variable length codes is particularly poor for symbols with probabilities greater than 0.5 as the best that can be achieved is to represent these symbols with a single-bit code.

Arithmetic coding provides a practical alternative to Huffman coding that can more closely approach theoretical maximum compression ratios [ix]. An arithmetic encoder converts a sequence of data symbols into a single fractional number and can approach the optimal fractional number of bits required to represent each symbol.

Example

Table 3.8 lists the five motion vector values (−2, −1, 0, 1, 2) and their probabilities from Example 1 in section 3.5.2.1. Each vector is assigned a **sub-range** within the range 0.0 to 1.0, depending on its probability of occurrence. In this example, (−2) has a probability of 0.1 and is given the subrange 0–0.1, i.e. the first 10 per cent of the total range 0 to 1.0. (−1) has a probability of 0.2 and is given the next 20 per cent of the total range, i.e. the sub-range 0.1–0.3.

After assigning a sub-range to each vector, the total range 0–1.0 has been divided amongst the data symbols (the vectors) according to their probabilities (Figure 3.49).

Table 3.8 Motion vectors, sequence 1: probabilities and sub-ranges

Vector	Probability	$\log_2(1/P)$	Sub-range
−2	0.1	3.32	0–0.1
−1	0.2	2.32	0.1–0.3
0	0.4	1.32	0.3–0.7
1	0.2	2.32	0.7–0.9
2	0.1	3.32	0.9–1.0

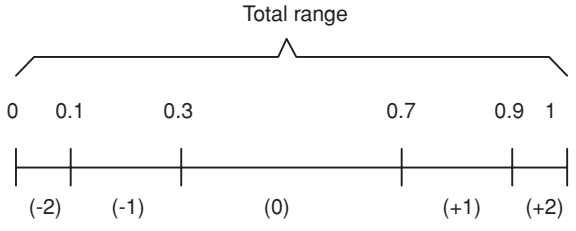


Figure 3.49 Sub-range example

Encoding procedure for vector sequence (0, −1, 0, 2).

Encoding procedure	Range (L → H)	Symbol	Sub-range (L → H)	Notes
1. Set the initial range	0 → 1.0			
2. For the first data symbol, find the corresponding sub-range (Low to High).		(0)	0.3 → 0.7	
3. Set the new range (1) to this sub-range	0.3 → 0.7			
4. For the next data symbol, find the sub-range L to H		(−1)	0.1 → 0.3	This is the sub-range within the interval 0 - 1
5. Set the new range (2) to this sub-range within the previous range	0.34 → 0.42			0.34 is 10% of the range; 0.42 is 30% of the range
6. Find the next sub-range		(0)	0.3 → 0.7	
7. Set the new range (3) within the previous range	0.364 → 0.396			0.364 is 30% of the range; 0.396 is 70% of the range
8. Find the next sub-range		(2)	0.9 → 1.0	
9. Set the new range (4) within the previous range	0.3928 → 0.396			0.3928 is 90% of the range; 0.396 is 100% of the range

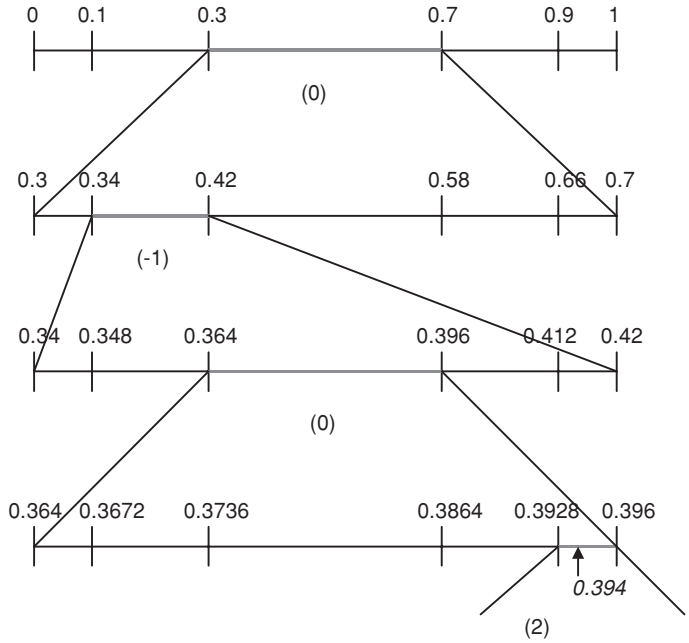


Figure 3.50 Arithmetic coding example

Each time a symbol is encoded, the range (L to H) becomes progressively smaller. At the end of the encoding process, four steps in this example, we are left with a final range (L to H). The entire sequence of data symbols can be represented by transmitting any fractional number that lies within this final range. In the example above, we could send any number in the range 0.3928 to 0.396: for example, 0.394. Figure 3.50 shows how the initial range (0, 1) is progressively partitioned into smaller ranges as each data symbol is processed. After encoding the first symbol, vector 0, the new range is (0.3, 0.7). The next symbol (vector –1) selects the sub-range (0.34, 0.42) which becomes the new range, and so on. The final symbol, vector +2, selects the sub-range (0.3928, 0.396) and the number 0.394 falling within this range is transmitted. 0.394 can be represented as a fixed-point fractional number using 9 bits, so our data sequence (0, −1, 0, 2) is compressed to a 9-bit quantity.

Decoding procedure

Decoding procedure	Range	Sub-range	Decoded symbol
1. Set the initial range	0 → 1		
2. Find the sub-range in which the received number falls. This indicates the first data symbol.		0.3 → 0.7	(0)
3. Set the new range (1) to this sub-range	0.3 → 0.7		

(*Continued*)

Decoding procedure	Range	Sub-range	Decoded symbol
4. Find the sub-range **of the new range** in which the received number falls. This indicates the second data symbol.		0.34 → 0.42	(−1)
5. Set the new range (2) to this sub-range within the previous range	0.34 → 0.42		
6. Find the sub-range in which the received number falls and decode the third data symbol.		0.364 → 0.396	(0)
7. Set the new range (3) to this sub-range within the previous range	0.364 → 0.396		
8. Find the sub-range in which the received number falls and decode the fourth data symbol.		0.3928 → 0.396	

The principal advantage of arithmetic coding is that the transmitted number, 0.394 in this case, which may be represented as a fixed-point number with sufficient accuracy using 9 bits, is not constrained to an integral number of bits for each transmitted data symbol. To achieve optimal compression, the sequence of data symbols should be represented with:

$$\log_2(1/P_0) + \log_2(1/P_{-1}) + \log_2(1/P_0) + \log_2(1/P_2) \text{ bits} = 8.28 \text{ bits}$$

In this example, arithmetic coding achieves 9 bits, which is close to optimum. A scheme using an integral number of bits for each data symbol such as Huffman coding is unlikely to come so close to the optimum number of bits and in general, arithmetic coding can out-perform Huffman coding.

3.5.3.1 Context-based Arithmetic Coding

Successful entropy coding depends on accurate models of symbol probability. Context-based Arithmetic Encoding (CAE) uses local spatial and/or temporal characteristics to estimate the probability of a symbol to be encoded. CAE is used in the JBIG standard for bi-level image compression [x] and has been adopted for coding binary shape 'masks' in MPEG-4 Visual and entropy coding in certain Profiles of H.264 (Chapter 7).

3.6 The hybrid DPCM/DCT video CODEC model

The major video coding standards released since the early 1990s have been based on the same generic design or model of a video CODEC that incorporates a motion estimation and compensation first stage, sometimes described as DPCM, a transform stage and an entropy encoder. The model is often described as a hybrid DPCM/DCT CODEC. Any CODEC that is compatible with H.261, H.263, MPEG-1, MPEG-2, MPEG-4 Visual, H.264/AVC or VC-1 has to implement a similar set of basic coding and decoding functions, although there are many differences of detail between the standards and between implementations.

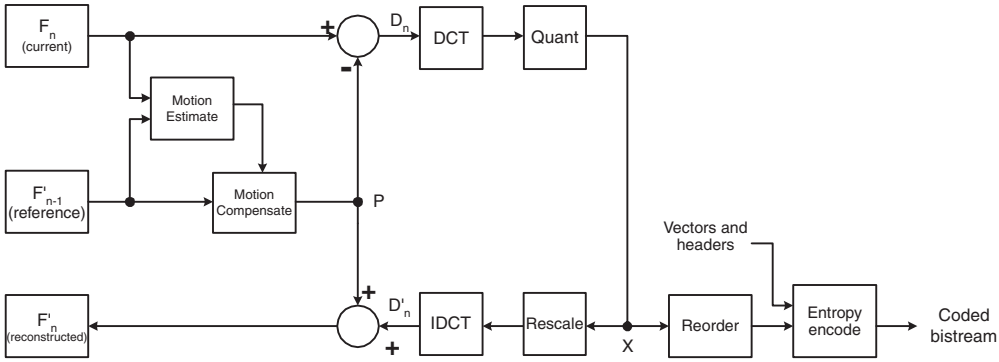


Figure 3.51 DPCM/DCT video encoder

Figure 3.51 and Figure 3.52 show the basic DPCM/DCT hybrid encoder and decoder. In the encoder, video frame n (F_n) is processed to produce a coded or compressed bitstream. In the decoder, the compressed bitstream, shown at the right of the figure, is decoded to produce a reconstructed video frame F'_n. The reconstructed output frame is not usually identical to the source frame. The figures have been deliberately drawn to highlight the common elements within encoder and decoder. Most of the functions of the decoder are actually contained within the encoder, the reason for which will be explained later.

Encoder data flow

There are two main data flow paths in the encoder, left to right (encoding) and right to left (reconstruction). The encoding flow is as follows:

1. An input video frame F_n is presented for encoding and is processed in units of a macroblock, corresponding to a 16 × 16 pixel region of the video image.
2. F_n is compared with a **reference** frame, for example the previous encoded frame (F'_{n-1}). A motion estimation function finds a 16 × 16 region in F'_{n-1} that 'matches' the current macroblock in F_n. The offset between the current macroblock position and the chosen reference region is a motion vector MV.
3. Based on the chosen motion vector MV, a motion compensated prediction P is generated, the 16 × 16 region selected by the motion estimator. P may consist of interpolated sub-pixel data.
4. P is subtracted from the current macroblock to produce a residual or difference macroblock D.

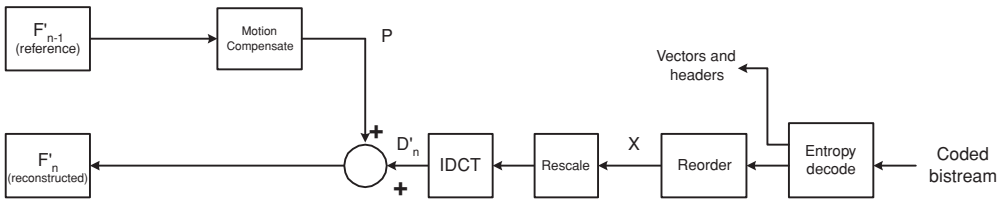


Figure 3.52 DPCM/DCT video decoder

5. D is transformed using the DCT. Typically, D is split into 8×8 or 4×4 sub-blocks and each sub-block is transformed separately.
6. Each sub-block is quantized (X).
7. The DCT coefficients of each sub-block are reordered and run-level coded.
8. Finally, the coefficients, motion vector and associated header information for each macroblock are entropy encoded to produce the compressed bitstream.

The reconstruction data flow is as follows:

1. Each quantized macroblock X is rescaled and inverse transformed to produce a decoded residual D'. Note that the non-reversible quantization process means that D' is not identical to D, i.e. distortion has been introduced.
2. The motion compensated prediction P is added to the residual D' to produce a reconstructed macroblock. The reconstructed macroblocks are combined to produce reconstructed frame F'_n.

After encoding a complete frame, the reconstructed frame F'_n may be used as a reference frame for the next encoded frame F_{n+1}.

Decoder data flow

1. A compressed bitstream is entropy decoded to extract coefficients, motion vector and header for each macroblock.
2. Run-level coding and reordering are reversed to produce a quantized, transformed macroblock X.
3. X is rescaled and inverse transformed to produce a decoded residual D'.
4. The decoded motion vector is used to locate a 16×16 region in the decoder's copy of the previous (reference) frame F'_{n-1}. This region becomes the motion compensated prediction P.
5. P is added to D' to produce a reconstructed macroblock. The reconstructed macroblocks are saved to produce decoded frame F'_n.

After a complete frame is decoded, F'_n is ready to be displayed and may also be stored as a reference frame for the next decoded frame F'_{n+1}.

It is clear from the figures and from the above explanation that the encoder includes a decoding path : rescale, IDCT, reconstruct. This is necessary to ensure that the encoder and decoder use identical reference frames F'_{n-1} for motion compensated prediction.

Example

A 25-Hz video sequence in CIF format, with 352×288 luminance samples and 176×144 red/blue chrominance samples per frame, is encoded and decoded using a DPCM/DCT CODEC. Figure 3.53 shows a CIF video frame (F_n) that is to be encoded and Figure 3.54 shows the reconstructed previous frame F'_{n-1}. Note that F'_{n-1} has been encoded and decoded and shows some distortion. The difference between F_n and F'_{n-1} **without** motion compensation (Figure 3.55) clearly still contains significant energy, especially around the edges of moving areas.

Figure 3.53 Input frame F_n

Figure 3.54 Reconstructed reference frame F'_{n-1}

Figure 3.55 Residual F_n-F'_{n-1} : no motion compensation

Motion estimation is carried out with a 16 × 16 block size and half-pixel accuracy, producing the set of vectors shown in Figure 3.56, superimposed on the current frame for clarity. Many of the vectors are zero and are shown as dots, which means that the best match for the current macroblock is in the same position in the reference frame. Around moving areas, the vectors point in the direction that blocks have moved **from**. E.g. the man on the left is walking to the left; the vectors therefore point to the **right**, i.e. where he has come from. Some of the vectors do not appear to correspond to 'real' movement, e.g. on the surface of the table, but indicate simply that the best match is not at the same position in the reference frame. 'Noisy' vectors like these often occur in homogeneous regions of the picture, where there are no clear object features in the reference frame.

The motion-compensated reference frame (Figure 3.57) is the reference frame 'reorganized' according to the motion vectors. For example, note that the walking person has been moved to the left to provide a better match for the same person in the current frame and that the hand of the left-most person has been moved down to provide an improved match. Subtracting the motion compensated reference frame from the current frame gives the motion-compensated residual in Figure 3.58 in which the energy has clearly been reduced, particularly around moving areas.

Figure 3.59 shows a macroblock from the original frame, taken from around the head of the figure on the right, and Figure 3.60 the luminance residual after motion compensation. Applying a 2D DCT to the top-right 8 × 8 block of luminance samples (Table 3.9) produces the DCT coefficients listed in Table 3.10. The magnitude of each coefficient is plotted in Figure 3.61; note that the larger coefficients are clustered around the top-left (DC) coefficient.

Figure 3.56 16 × 16 motion vectors superimposed on frame

Figure 3.57 Motion compensated reference frame

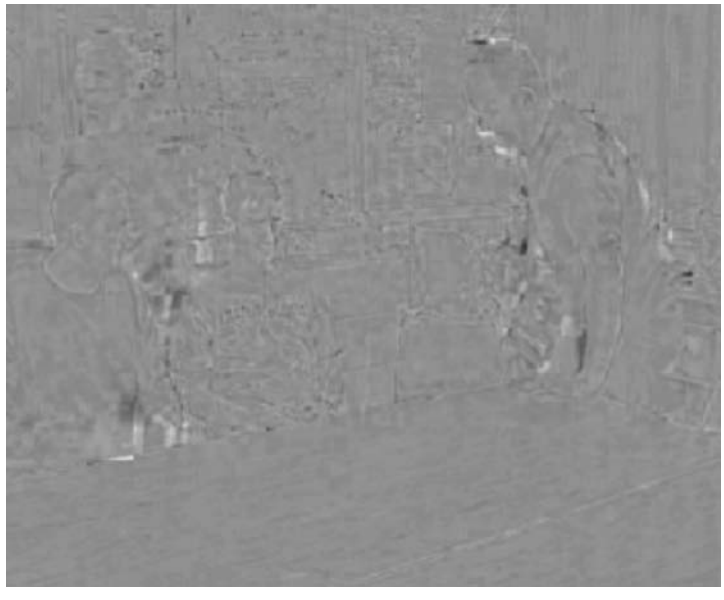


Figure 3.58 Motion compensated residual frame

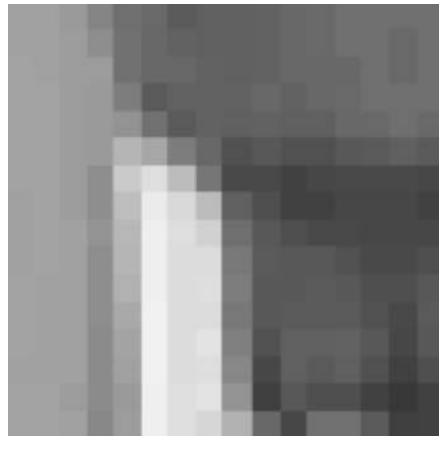


Figure 3.59 Original macroblock : luminance

Figure 3.60 Residual macroblock : luminance

Table 3.9 Residual luminance samples : top-right 8 × 8 block

−4	−4	−1	0	1	1	0	−2
1	2	3	2	−1	−3	−6	−3
6	6	4	−4	−9	−5	−6	−5
10	8	−1	−4	−6	−1	2	4
7	9	−5	−9	−3	0	8	13
0	3	−9	−12	−8	−9	−4	1
−1	4	−9	−13	−8	−16	−18	−13
14	13	−1	−6	3	−5	−12	−7

Table 3.10 DCT coefficients

−13.50	20.47	20.20	2.14	−0.50	−10.48	−3.50	−0.62
10.93	−11.58	−10.29	−5.17	−2.96	10.44	4.96	−1.26
−8.75	9.22	−17.19	2.26	3.83	−2.45	1.77	1.89
−7.10	−17.54	1.24	−0.91	0.47	−0.37	−3.55	0.88
19.00	−7.20	4.08	5.31	0.50	0.18	−0.61	0.40
−13.06	3.12	−2.04	−0.17	−1.19	1.57	−0.08	−0.51
1.73	−0.69	1.77	0.78	−1.86	1.47	1.19	0.42
−1.99	−0.05	1.24	−0.48	−1.86	−1.17	−0.21	0.92

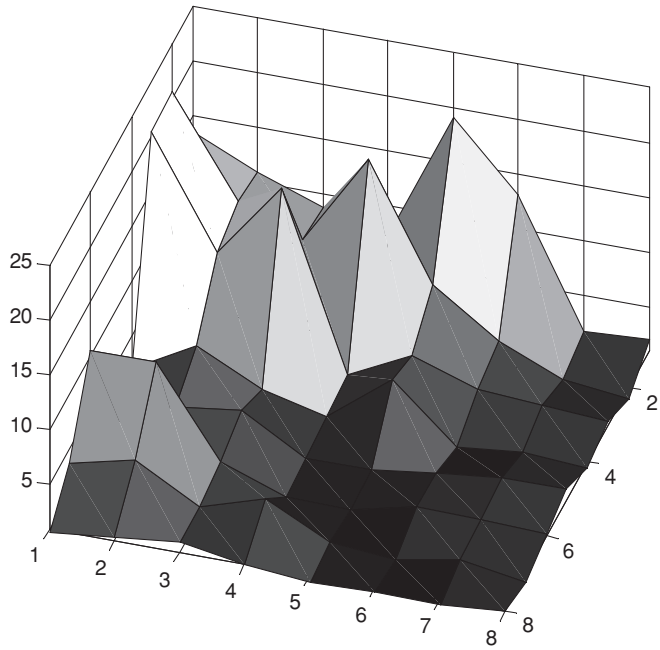


Figure 3.61 DCT coefficient magnitudes : top-right 8 × 8 block

A simple forward quantizer is applied:

$$\text{Qcoeff} = \text{round}(\text{coeff}/\text{Qstep})$$

where Qstep is the quantizer step size, 12 in this example. Small-valued coefficients become zero in the quantized block (Table 3.11) and the non-zero outputs are clustered around the top-left (DC) coefficient.

The quantized block is re-ordered in a zigzag scan starting at the top-left to produce a linear array:

$$-1,\ 2,\ 1,\ -1,\ -1,\ 2,\ 0,\ -1,\ 1,\ -1,\ 2,\ -1,\ -1,\ 0,$$
$$0,\ -1,\ 0,\ 0,\ 0,\ -1,\ -1,\ 0,\ 0,\ 0,\ 0,\ 0,\ 1,\ 0,\ \ldots$$

Table 3.11 Quantized coefficients

−1	2	2	0	0	−1	0	0
1	−1	−1	0	0	1	0	0
−1	1	−1	0	0	0	0	0
−1	−1	0	0	0	0	0	0
2	−1	0	0	0	0	0	0
−1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Table 3.12 Variable length coding example

Run, Level, Last	VLC including sign bit
(0,−1, 0)	101
(0,2, 0)	11100
(0,1, 0)	100
(0,−1, 0)	101
(0,−1, 0)	101
(0,2, 0)	11100
(1,−1, 0)	1101
...	...
(5,1, 1)	00100110

This array is processed to produce a series of (zero run, level) pairs:

$$(0,\ -1)(0,\ 2)(0,\ 1)(0,\ -1)(0,\ -1)(0,\ 2)(1,\ -1)(0,\ 1)(0,\ -1)$$
$$(0,\ 2)(0,\ -1)(0,\ -1)(2,\ -1)(3,\ -1)(0,\ -1)(5,\ 1)(\text{EOB})$$

'EOB' (End Of Block) indicates that the remainder of the coefficients are zero.

Each (run, level) pair is encoded as a VLC. Using the MPEG-4 Visual TCOEF table (Table 3.6), the VLCs shown in Table 3.12 are produced.

The final VLC signals that LAST=1, indicating that this is the end of the block. The motion vector for this macroblock is (0, 1), i.e. the vector points downwards. The predicted vector based on neighbouring macroblocks is (0,0) and so the motion vector difference values are MVDx=0, MVDy=+1. Using the MPEG4 MVD table (Table 3.7), these are coded as (1) and (0010) respectively.

The macroblock is transmitted as a series of VLCs, including a macroblock header, motion vector difference (X, Y) and transform coefficients (TCOEF) for each 8×8 block.

At the decoder, the VLC sequence is decoded to extract header parameters, MVDx and MVDy and (run,level) pairs for each block. The 64-element array of reordered coefficients is reconstructed by inserting (run) zeros before every (level). The array is then ordered to produce an 8×8 block identical to Table 3.11. The quantized coefficients are rescaled using:

$$\text{Rcoeff} = \text{Qstep.Qcoeff}$$

Where Qstep=12 as before, to produce the block of coefficients shown in Table 3.13. Note that the block is significantly different from the original DCT coefficients (Table 3.10) due to

Table 3.13 Rescaled coefficients

−12	24	24	0	0	−12	0	0
12	−12	−12	0	0	12	0	0
−12	12	−12	0	0	0	0	0
−12	−12	0	0	0	0	0	0
24	−12	0	0	0	0	0	0
−12	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Table 3.14 Decoded residual luminance samples

−3	−3	−1	1	−1	−1	−1	−3
5	3	2	0	−3	−4	−5	−6
9	6	1	−3	−5	−6	−5	−4
9	8	1	−4	−1	1	4	10
7	8	−1	−6	−1	2	5	14
2	3	−8	−15	−11	−11	−11	−2
2	5	−7	−17	−13	−16	−20	−11
12	16	3	−6	−1	−6	−11	−3

the quantization process. An Inverse DCT is applied to create a decoded residual block (Table 3.14) which is similar but not identical to the original residual block (Table 3.9). The original and decoded residual blocks are plotted side by side in Figure 3.62 and it is clear that the decoded block has less high-frequency variation because of the loss of high-frequency DCT coefficients through quantization.

The decoder forms its own predicted motion vector based on previously decoded vectors and recreates the original motion vector (0, 1). Using this vector, together with its own copy of the previously decoded frame F'_{n-1}, the decoder reconstructs the macroblock. The complete decoded frame is shown in Figure 3.63. Because of the quantization process, some distortion has been introduced, for example around detailed areas such as the faces and the equations on the whiteboard and there are some obvious edges along 8 × 8 block boundaries. The complete sequence was compressed by around 300 times, i.e. the coded sequence occupies less than 1/300 the size of the uncompressed video, and so significant compression was achieved at the expense of relatively poor image quality.

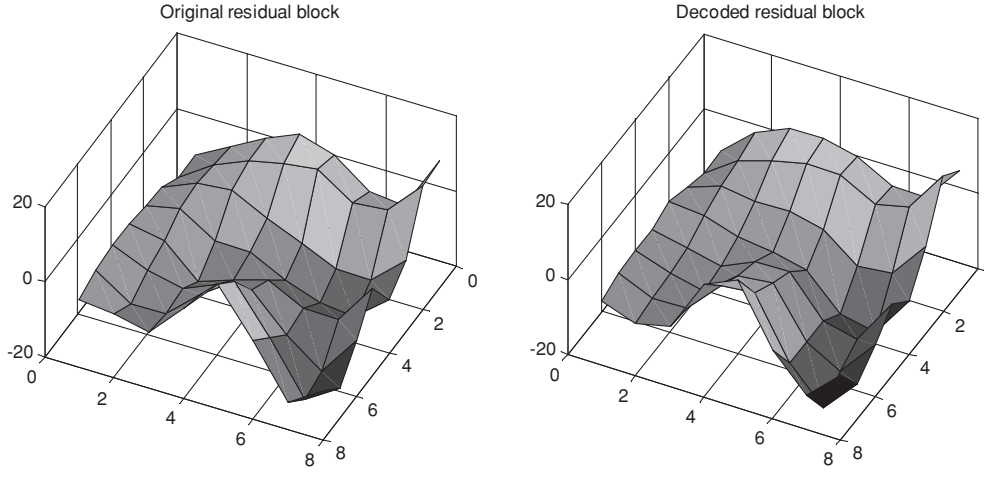


Figure 3.62 Comparison of original and decoded residual blocks

Figure 3.63 Decoded frame F'_n

3.7 Summary

The video coding tools described in this chapter, motion compensated prediction, transform coding, quantization and entropy coding, form the basis of the reliable and effective coding model that has dominated the field of video compression for over 10 years. This coding model is at the heart of the H.264/AVC standard. The next chapter introduces the main features of H.264/AVC and the standard is discussed in detail in following chapters.

3.8 References

i. Information technology – lossless and near-lossless compression of continuous-tone still images: Baseline, ISO/IEC 14495-1:2000 ('JPEG-LS').
ii. B. Horn and B. G. Schunk, 'Determining Optical Flow', *Artificial Intelligence*, 17:185–203, 1981.
iii. T. Wedi, 'Adaptive Interpolation Filters and High Resolution Displacements for Video Coding', *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 16, no. 4, pp 484–491, April 2006.
iv. K. R. Rao and P. Yip, *'Discrete Cosine Transform*. Academic Press, 1990.
v. S. Mallat, *A Wavelet Tour of Signal Processing*. Academic Press, 1999.
vi. N. Nasrabadi and R. King, 'Image coding using vector quantization: a review', *IEEE Trans. Communications*, vol. 36, no. 8, August 1988.
vii. W. A. Pearlman, 'Trends of tree-based, set-partitioned compression techniques in still and moving image systems', Proc. International Picture Coding Symposium, Seoul, April 2001.
viii. D. Huffman, 'A method for the construction of minimum redundancy codes', Proceedings of the IRE, vol. 40, pp.1098–1101, 1952.
ix. I. Witten, R. Neal and J. Cleary, 'Arithmetic coding for data compression', Communications of the ACM, 30(6), June 1987.
x. Information technology – coded representation of picture and audio information – progressive bi-level image compression, ITU-T Rec. T.82 ('JBIG').

4

What is H.264?

4.1 Introduction

This chapter introduces H.264 from a number of perspectives. First, we address the question – what is H.264? H.264 can mean different things from different viewpoints. It is an industry standard; it defines a format for compressed video data; it provides a set of tools that can be used in a variety of ways to compress and communicate visual information; it is a stage in an evolving series of standardized methods for video compression. We look at the basic operations carried out within an H.264 encoder (compressor) and decoder (decompressor). We look in more depth inside H.264, examining what is contained within the H.264 standard, what are the main sets of tools and components, presenting an outline of the H.264 compression syntax or format and looking in more detail at the flow of data through a typical H.264 codec. Finally we discuss some of the practical issues – just how good is H.264 at doing its job; how is H.264 used in 'real' applications.

4.2 What is H.264?

H.264 Advanced Video Coding is an industry standard for video coding, but it is also a popular format for coded video, a set of tools for video compression and a stage in a continuously evolving digital video communication landscape. This section introduces some of these 'views' of H.264/AVC.

4.2.1 A video compression format

H.264 is a method and format for video compression, the process of converting digital video into a format that takes up less capacity when it is stored or transmitted. Video compression or video coding is an essential technology for applications such as digital television, DVD-Video, mobile TV, videoconferencing and internet video streaming. Standardizing video compression makes it possible for products from different manufacturers such as encoders, decoders and storage media to inter-operate. An **encoder** converts video into a compressed format and a **decoder** converts compressed video back into an uncompressed format.

The H.264 Advanced Video Compression Standard, Second Edition Iain E. Richardson
© 2010 John Wiley & Sons, Ltd

In a typical application of H.264 such as remote surveillance, video from a camera is encoded or compressed using H.264 to produce an H.264 bitstream. This is sent across a network to a decoder which reconstructs a version of the source video, Figure 1.1. Figure 1.1 shows how source video, uncompressed video material, is encoded by an encoder and stored on a server. The compressed video, a sequence of bits, is transmitted to one or more clients. This example shows clients with widely varying display capabilities from a full-sized TV display to a mobile screen and this highlights a current challenge in video coding – how to cater for receivers with different capabilities. In a two-way 'conversational' video application such as videoconferencing, Apple iChat, Skype Video, etc, both ends of the system require an encoder to compress video from the local camera and a decoder to decompress video from the remote camera, Figure 1.2.

4.2.2 An industry standard

Recommendation H.264: Advanced Video Coding is a document co-published by two international standards bodies, the ITU-T (International Telecommunication Union) and the ISO/IEC (International Organisation for Standardisation/International Electrotechnical Commission) [i]. It defines a format or syntax for compressed video and a method for decoding this syntax to produce a displayable video sequence. The standard document does not actually specify how to encode digital video – this is left to the manufacturer of a video encoder – but in practice the encoder is likely to mirror the steps of the decoding process. Figure 4.1 shows the encoding and decoding processes and highlights the parts that are covered by the H.264 standard.

The H.264/AVC standard was first published in 2003, with several revisions and updates published since then. It builds on the concepts of earlier standards such as MPEG-2 and MPEG-4 Visual and offers the potential for better compression efficiency, i.e. better-quality compressed video, and greater flexibility in compressing, transmitting and storing video.

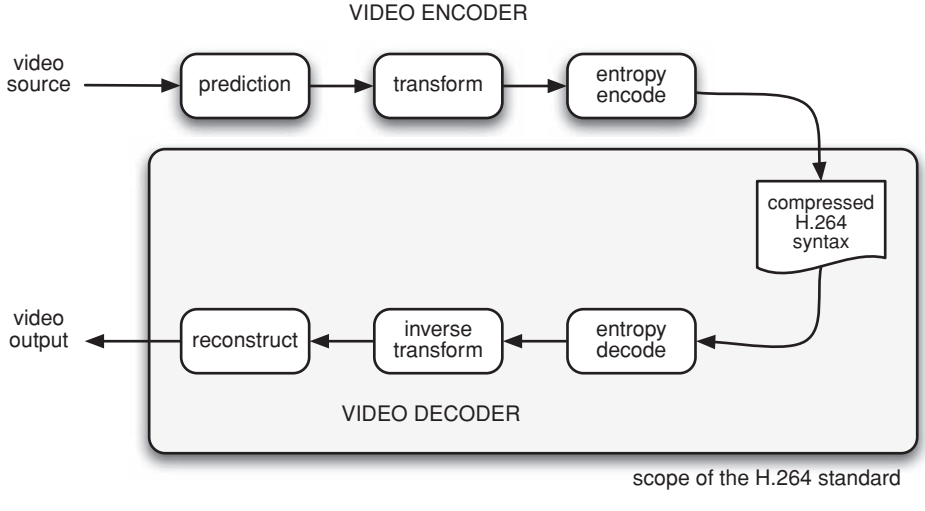


Figure 4.1 The H.264 video coding and decoding process

4.2.3 A toolkit for video compression

H.264/AVC describes a set of tools or methods for video compression. The standard specifies how video coded with these tools should be represented and decoded. A video encoder may choose which tools to use and how to apply them to the current video sequence, with some constraints. An H.264-compliant decoder must be capable of using a defined sub-set of tools, known as a **profile**.

4.2.4 Better video compression

One of the most important drivers for the standardization of H.264 and its subsequent adoption by industry is its improved performance compared with earlier standards. The benchmark for mass-market applications such as digital TV and consumer video storage on DVD-Video is the earlier MPEG-2 standard [ii]. H.264 offers significantly better compression performance than MPEG-2 Visual. Using H.264 it is possible to compress video into a much smaller number of bits than using MPEG-2, for the same video resolution and image quality. This means, for example, that much more video material can be stored on a disk or transmitted over a broadcast channel by using the H.264 format.

4.3 How does an H.264 codec work?

An H.264 video encoder carries out prediction, transforming and encoding processes (Figure 4.1) to produce a compressed H.264 bitstream. An H.264 video decoder carries out the complementary processes of decoding, inverse transform and reconstruction to produce a decoded video sequence. As Figure 4.2 shows, a sequence of original video frames or fields is

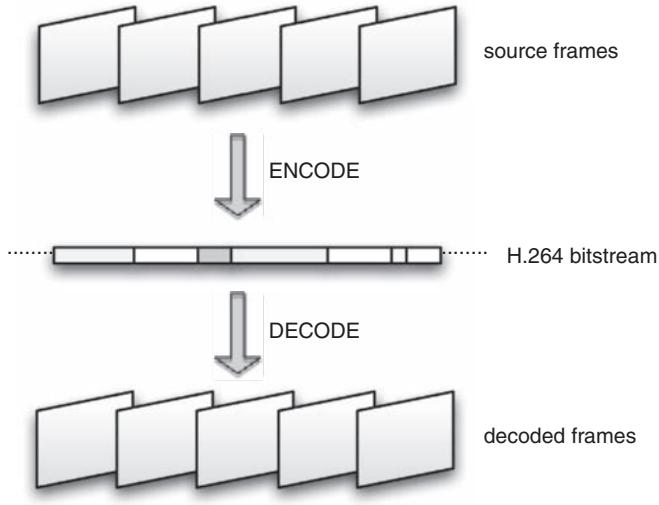


Figure 4.2 Video coding: source frames, encoded bitstream, decoded frames

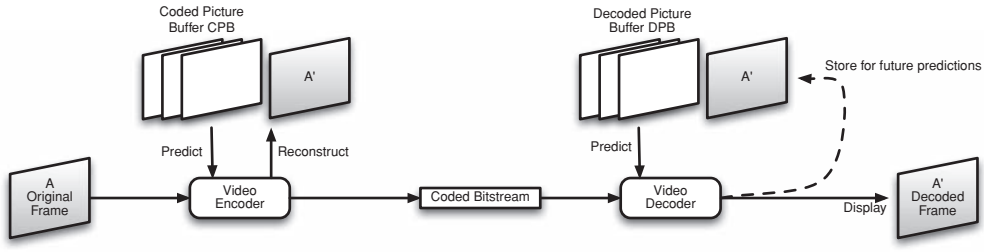


Figure 4.3 Video codec: high level view

encoded into the H.264 format, a series of bits that represents the video in compressed form. This compressed bitstream is stored or transmitted and can be decoded to reconstruct the video sequence. The decoded version is, in general, not identical to the original sequence because H.264 is a lossy compression format, i.e. some picture quality is lost during compression.

A frame or field to be coded, e.g. Frame A in Figure 4.3, is processed by an H.264-compatible video encoder. As well as coding and sending the frame as part of the coded bitstream or coded file, the encoder reconstructs the frame, i.e. creates a copy of the decoded frame A' that will eventually be produced by the decoder. This reconstructed copy may be stored in a coded picture buffer, CPB, and used during the encoding of further frames. The decoder receives the coded bitstream and decodes frame A' for display or further processing. At the same time, the decoder may store a copy of frame A' in a decoded picture buffer, DPB, to be used during the decoding of further frames.

The structure of a typical H.264 codec is shown in Figure 4.4 (encoder) and Figure 4.5 (decoder) in slightly more detail. Data is processed in units of a **macroblock** (MB) corresponding to 16 × 16 displayed pixels. In the encoder, a prediction macroblock is generated and subtracted from the current macroblock to form a residual macroblock; this is transformed, quantized and encoded. In parallel, the quantized data are re-scaled and inverse transformed and added to the prediction macroblock to reconstruct a coded version of the frame which is stored for later predictions. In the decoder, a macroblock is decoded, re-scaled and inverse transformed to form a decoded residual macroblock. The decoder generates the same

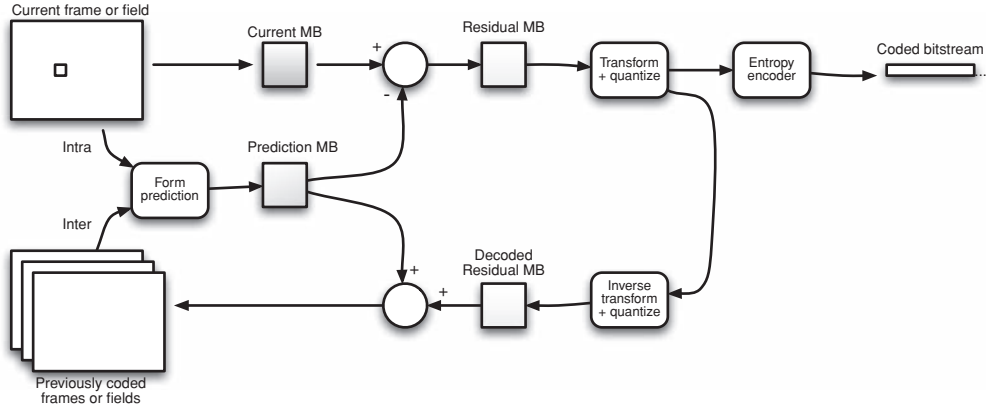


Figure 4.4 Typical H.264 encoder

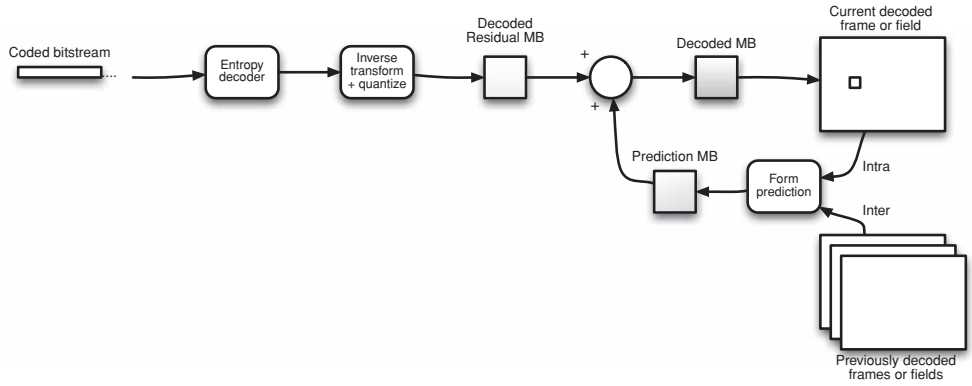


Figure 4.5 Typical H.264 decoder

prediction that was created at the encoder and adds this to the residual to produce a decoded macroblock.

4.3.1 Encoder processes

4.3.1.1 Prediction

The encoder forms a prediction of the current macroblock based on previously-coded data, either from the current frame using **intra** prediction or from other frames that have already been coded and transmitted using **inter** prediction. The encoder subtracts the prediction from the current macroblock to form a **residual**[1] (Figure 4.6).

The prediction methods supported by H.264 are more flexible than those in previous standards, enabling accurate predictions and hence efficient video compression. Intra prediction uses 16 × 16 and 4 × 4 block sizes to predict the macroblock from surrounding, previously coded pixels within the same frame (Figure 4.7).

The values of the previously-coded neighbouring pixels are extrapolated to form a prediction of the current macroblock. Figure 4.8 shows an example. A 16 × 16 prediction block is formed, an approximation of the original macroblock. Subtracting the prediction from the original macroblock produces a residual block (also containing 16 × 16 samples).

Inter prediction uses a range of block sizes from 16 × 16 down to 4 × 4 to predict pixels in the current frame from similar regions in previously coded frames (Figure 4.9). These previously coded frames may occur before or after the current frame in display order. In the example shown in Figure 4.9, macroblock 1 (MB1) in the current frame is predicted from a 16 × 16 region in the most recent 'past' frame. MB2 is predicted from two previously coded frames. The upper 8 × 16 block of samples, a 'partition', is predicted from a past frame and the lower 8 × 16 partition is predicted from a future frame.

[1] Finding a suitable inter prediction is often described as **motion estimation**. Subtracting an inter prediction from the current macroblock is **motion compensation**.

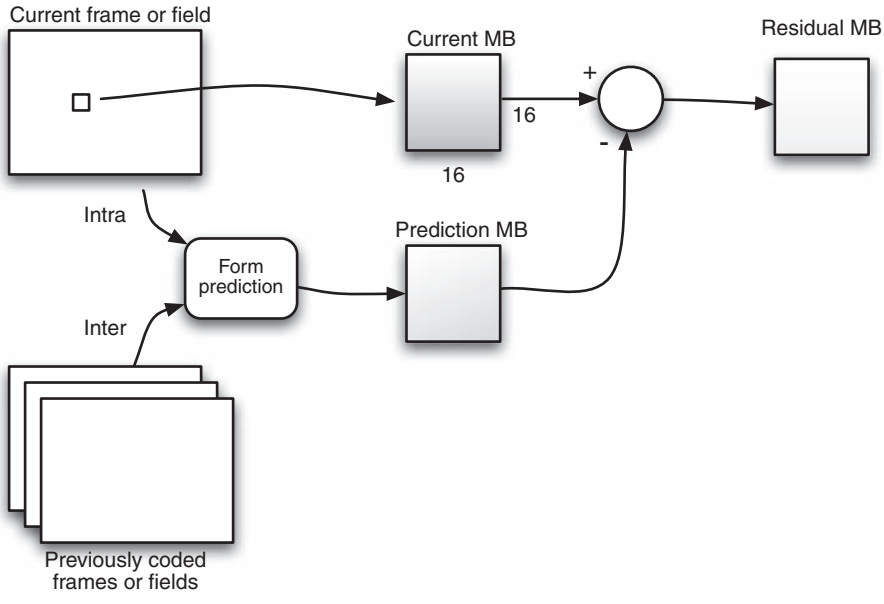


Figure 4.6 Prediction: flow diagram

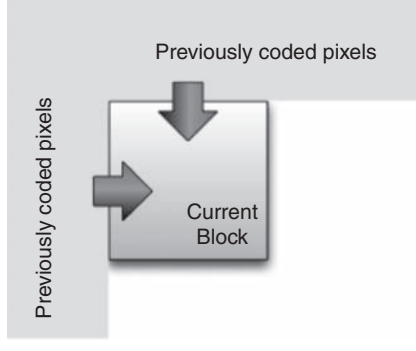


Figure 4.7 Intra prediction

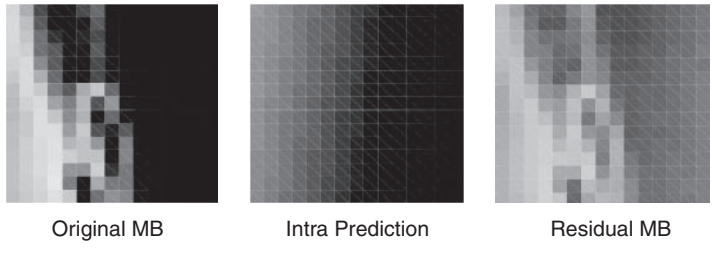


Figure 4.8 Original macroblock, intra prediction and residual

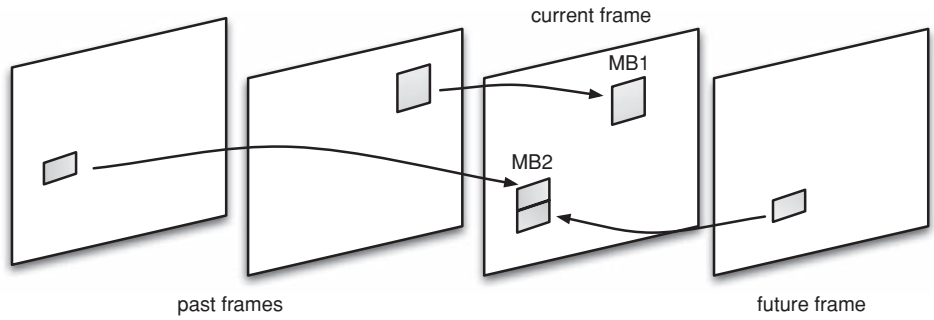


Figure 4.9 Inter prediction

Figure 4.10 shows an example of an original macroblock and its prediction from the previous frame. In this example, the 16 × 16 prediction block is a good match for the current macroblock. The values in the residual macroblock are mostly near zero.

4.3.1.2 Transform and quantization

A block of residual samples is transformed using a 4 × 4 or 8 × 8 **integer transform**, an approximate form of the Discrete Cosine Transform (**DCT**). The transform outputs a set of **coefficients**, each of which is a weighting value for a standard basis pattern. When combined, the weighted basis patterns re-create the block of residual samples (Figure 4.11).

The output of the transform, a block of transform coefficients, is **quantized**, i.e. each coefficient is divided by an integer value. Quantization reduces the precision of the transform coefficients according to a quantization parameter (QP). For example, the original coefficient values in Figure 4.12 are divided by a QP or step size of 8 and rounded to the nearest integer. Typically, the result is a block in which most or all of the coefficients are zero, with a few non-zero coefficients. Setting QP to a high value means that more coefficients are set to zero, resulting in high compression at the expense of poor decoded image quality. Setting QP to a low value means that more non-zero coefficients remain after quantization, resulting in better image quality at the decoder but also in lower compression (Figure 4.13).

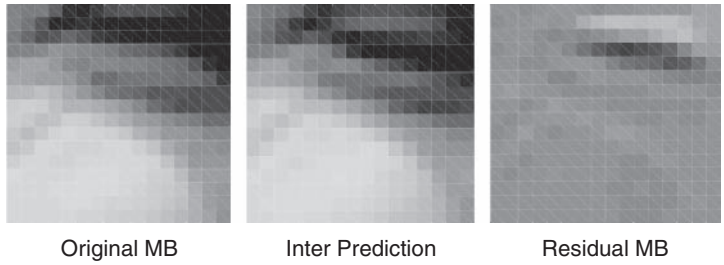


Figure 4.10 Original macroblock, inter prediction and residual

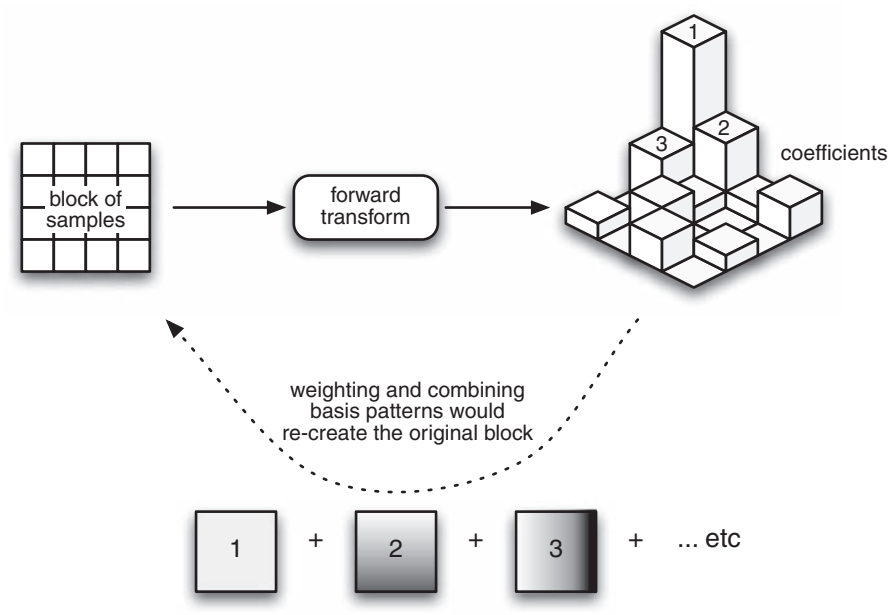


Figure 4.11 Forward transform

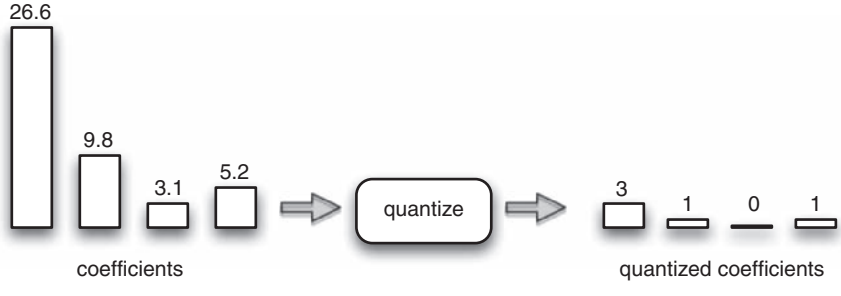


Figure 4.13 Quantization example

73 87 64 13 40 63 23 2 36 24 68 26 29 98 67 12 **Original block (4x4)**	181.3 47.4 –65.8 4.0 5.7 29.6 16.4 –2.2 40.3 18.3 –28.8 –13.8 13.1 –20.2 13.8 33.9 **Forward transform coefficients**
18 5 –7 0 1 3 2 0 4 2 –3 –1 1 –2 1 3 **Quantized coefficients (step size = 10)**	9 2 –3 0 0 1 1 0 2 1 –1 –1 1 –1 1 2 **Quantized coefficients (step size = 20)**

Figure 4.13 Example: Block, transform coefficients, quantized coefficients

4.3.1.3 Bitstream encoding

The video coding process produces a number of values that must be **encoded** to form the compressed bitstream. These values include:

- Quantized transform coefficients
- Information to enable the decoder to re-create the prediction
- Information about the structure of the compressed data and the compression tools used during encoding
- Information about the complete video sequence.

These values and parameters, **syntax elements**, are converted into binary codes using **variable length coding** and/or **arithmetic coding**. Each of these encoding methods produces an efficient, compact binary representation of the information. The encoded bitstream can then be stored and/or transmitted.

4.3.2 Decoder processes

4.3.2.1 Bitstream decoding

A video decoder receives the compressed H.264 bitstream, decodes each of the syntax elements and extracts the information described above, i.e. quantized transform coefficients, prediction information, etc. This information is then used to reverse the coding process and recreate a sequence of video images.

4.3.2.2 Rescaling and inverse transform

The quantized transform coefficients are **re-scaled**. Each coefficient is multiplied by an integer value to restore its original scale.[2] In the example of Figure 4.14, the quantized coefficients are each multiplied by a QP or step size of 8. The re-scaled coefficients are similar but not identical to the originals (Figure 4.12).

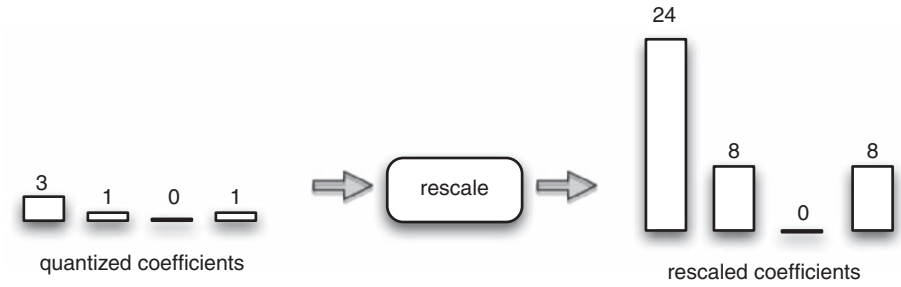


Figure 4.14 Rescaling example

[2] This is often described as **inverse quantization** but it is important to note that quantization is not a fully reversible process. Information removed during quantization cannot be restored during re-scaling.

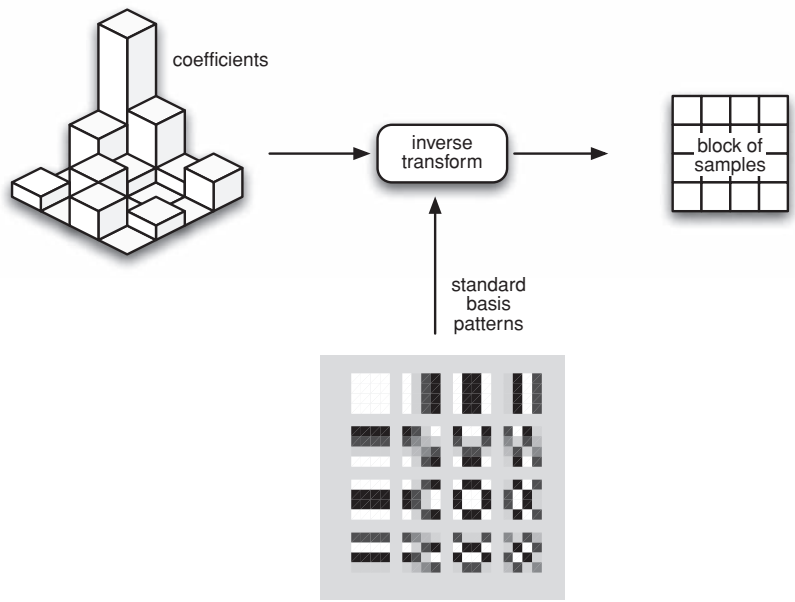


Figure 4.15 Inverse transform: combining weighted basis patterns to create a 4 × 4 image block

An inverse transform combines the standard basis patterns, weighted by the re-scaled coefficients, to re-create each block of residual data. Figure 4.15 shows how the inverse DCT or integer transform creates an image block by weighting each basis pattern according to a coefficient value and combining the weighted basis patterns. These blocks are combined together to form a residual macroblock. In Figure 4.16, the quantized coefficients of Figure 4.13 are rescaled using a quantization step size of 10 or 20 and inverse transformed. The reconstructed blocks are similar but not identical to the original block of Figure 4.13. The difference or loss is due to the forward quantization process. A larger quantization step size tends to produce a larger difference between original and reconstructed blocks.

180.0	50.0	–70.0	0.0	180.0	40.0	–60.0	0.0
10.0	30.0	20.0	0.0	0.0	20.0	20.0	0.0
40.0	20.0	–30.0	–10.0	40.0	20.0	–20.0	–20.0
10.0	–20.0	10.0	30.0	20.0	–20.0	20.0	40.0
Rescaled coefficients (QStep=10)				**Rescaled coefficients (QStep=20)**			
74	89	65	11	71	78	59	24
42	64	23	4	29	61	23	2
30	29	64	24	40	23	73	29
27	99	67	8	30	103	60	15
Inverse transform output (QStep=10)				**Inverse transform output (QStep=20)**			

Figure 4.16 Rescaled coefficients and inverse transform output

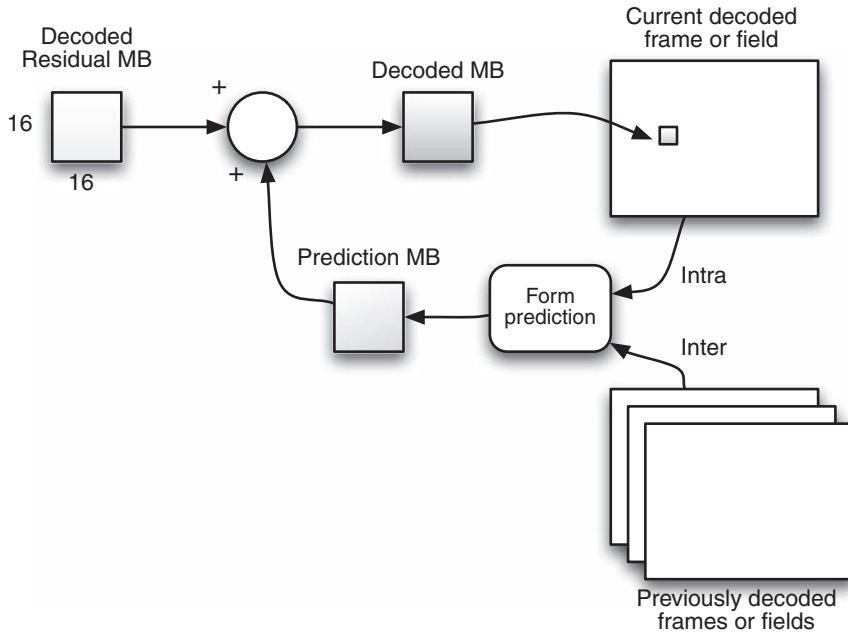


Figure 4.17 Reconstruction flow diagram

4.3.2.3 Reconstruction

For each macroblock, the decoder forms an identical prediction to the one created by the encoder using inter prediction from previously-decoded frames or intra prediction from previously-decoded samples in the current frame. The decoder adds the prediction to the decoded residual to **reconstruct** a decoded macroblock which can then be displayed as part of a video frame (Figure 4.17).

4.4 The H.264/AVC Standard

The current version of Recommendation H.264 [i] is a document of over 550 pages. It contains normative content, essential instructions that must be observed by H.264 codecs, and informative content, guidance that is not mandatory, and is organised as shown in Table 4.1. Note that as amendments or updated versions of the standard are published, further sections or Annexes may be added.

An H.264-compliant bitstream, a coded video sequence, should conform to the syntax definition, with syntax elements constructed according to the semantics described in Chapter 7 of the standard. An H.264 decoder should follow the parsing process in Chapter 9 of the standard to extract syntax elements from the bitstream and these should be decoded according to the decoding processes described in Chapter 8 of the standard in order to create a decoded video sequence.

Table 4.1 Overview of the H.264 standard document

Section	What it describes
0. Introduction	A brief overview of the purpose and main features of the standard.
1–5. Scope, References, Definitions, Abbreviations, Conventions	How H.264 fits with other published standards; terminology and conventions used throughout the document.
6. Data formats and relationships	Defines the basic formats assumed for video and coded data; principles of deriving relationships between coded units.
7. Syntax and semantics	A series of tables describing the syntax or bitstream format and an explanation of the meaning and allowed values of each syntax element (semantics)
8. Decoding process	Details of all the stages of processing required to decode a video sequence from H.264 syntax elements.
9. Parsing process	The processes required to extract syntax elements from a coded H.264 bitstream.
A. Profiles and levels	Profiles define subsets of video coding tools; levels define limits of decoder capabilities.
B. Byte stream format	Syntax and semantics of a stream of coded NAL units.
C. Hypothetical reference decoder	A hypothetical decoder 'model' that is used to determine performance limits.
D. Supplemental enhancement information	Information that may be sent in an H.264 bitstream that is not essential for decoding. Examples include user-generated data, buffering control, etc.
E. Video usability information	Non-essential information about display of the coded video.
G. Scalable Video Coding	A self-contained extension to the H.264/AVC standard that supports Scalable Video Coding (SVC). SVC supports efficient transmission of video at multiple spatial resolutions, frame rates and/or quality levels.

4.5 H.264 Profiles and Levels

H.264 supports a portfolio of 'tools'. These are algorithms or processes for coding and decoding video and related operations. These include tools essential to any implementation of H.264 such as the basic 4×4 transform, optional or interchangeable tools such as CABAC or CAVLC entropy coding, tools for specific applications or scenarios such as the SI/SP tools which can support streaming video and tools not directly related to video coding such as VUI and SEI messages.

An encoder has considerable flexibility in choosing and using tools defined in the standard. A decoder may be capable of operating with a very limited set of tools, for example if it is only required to decode bitstreams that use a subset of tools. H.264 **profiles** each define a specific subset of tools. An H.264 bitstream that conforms to a particular profile can only contain video coded using some or all of the tools within the profile; a profile-compliant decoder must be capable of decoding with every tool in the profile. The profiles therefore act as constraints on

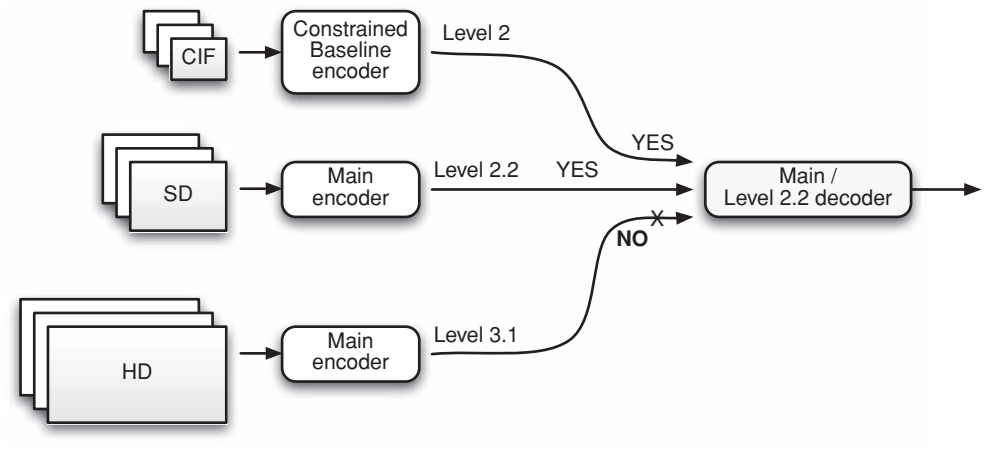


Figure 4.18 Profiles and Levels: example

the capabilities required by a decoder. The Main Profile is perhaps the most widely used at the present time, with tools that offer a good compromise between compression performance and computational complexity. The Constrained Baseline Profile is a subset of the Main Profile that is popular for low-complexity, low-delay applications such as mobile video.

A second constraint is the amount of data that a decoder can handle, which is likely to be limited by the processing speed, memory capacity and display size at the decoder. An H.264/AVC **level** specifies an upper limit on the frame size, processing rate (number of frames or blocks can be decoded per second) and working memory required to decode a video sequence. A particular decoder can only decode H.264 bitstreams up to a certain combination of Profile and Level.

Example

The decoder shown in Figure 4.18 can decode Main Profile bitstreams, up to Level 2.2 which corresponds to Standard Definition video coded at bitrates of 4Mbps or less. The figure shows three potential input bitstreams. The first is coded at CIF resolution with a maximum bitrate of 2Mbps (Level 2) using the tools in the Constrained Baseline Profile. Constrained Baseline is a subset of Main Profile and Level 2 is lower than Level 2.2, so the Main Profile/Level 2.2 decoder **can** decode this bitstream.

The second is coded at Standard Definition resolution, 720 × 576 displayed pixels, at a bitrate of less than 4Mbps (Level 2.2) by a Main Profile encoder. The decoder **can** decode this bitstream.

The third is coded at 720p High Definition resolution, 1280 × 720 displayed pixels, and is coded with a Main Profile encoder at Level 3.1. This is higher than the maximum Level supported by the decoder and so it **cannot** be decoded.

This example shows that Profiles and Levels are a useful mechanism for ensuring that a decoder's capabilities are not exceeded. The Profile and Level of an H.264 bitstream are signalled in the Sequence Parameter Set (Chapters 5 and 8).

4.6 The H.264 Syntax

H.264 provides a clearly defined format or **syntax** for representing compressed video and related information. Figure 4.19 shows an overview of the structure of the H.264 syntax.

At the top level, an H.264 **sequence** consists of a series of 'packets' or Network Adaptation Layer Units, NAL Units or NALUs. These can include **parameter sets** containing key parameters that are used by the decoder to correctly decode the video data and **slices**, which are coded video frames or parts of video frames.

At the next level, a **slice** represents all or part of a coded video frame and consists of a number of coded **macroblocks**, each containing compressed data corresponding to a 16 × 16 block of displayed pixels in a video frame. At the lowest level of Figure 4.19, a macroblock contains type information describing the particular choice of methods used to code the macroblock,

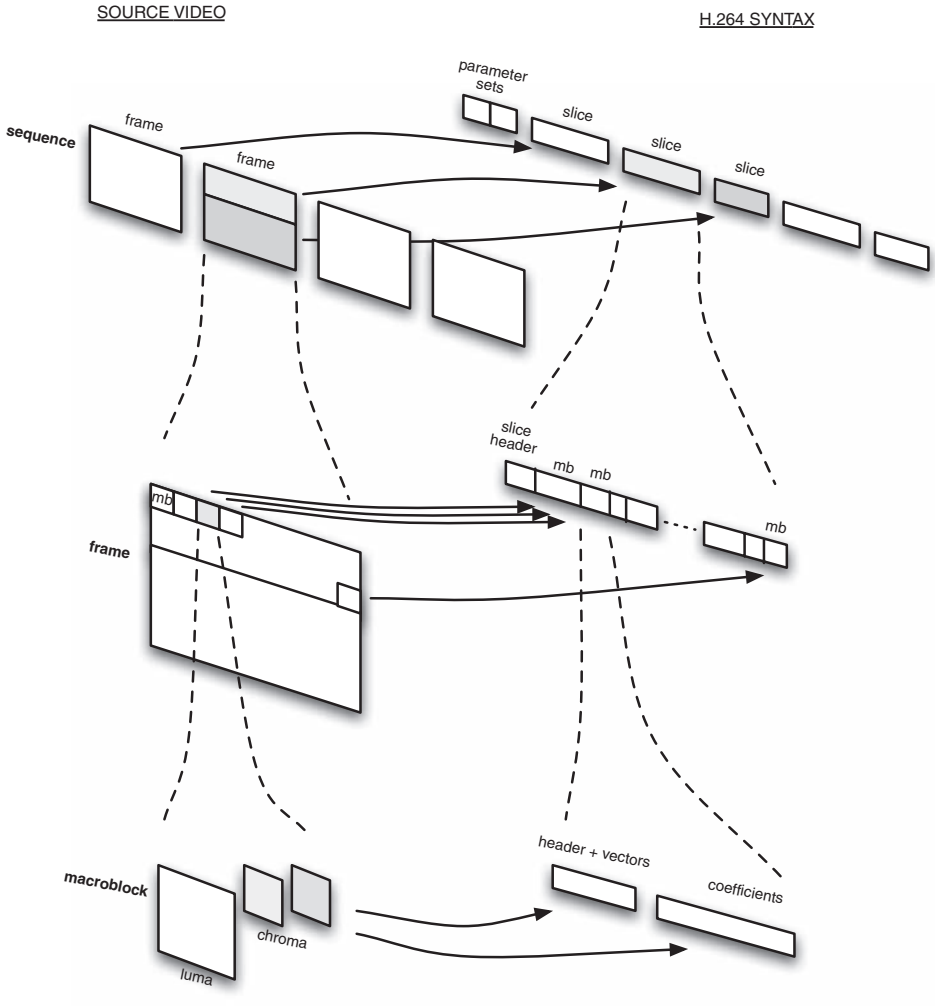


Figure 4.19 H.264 syntax: overview

prediction information such as coded motion vectors or intra prediction mode information and coded residual data.

Example

An H.264 decoder receives the following series of bits, describing an inter-coded macroblock or P MB coded using context-adaptive VLCs:

1101111100011100011110010 (alternate syntax elements highlighted).

Figure 4.20 shows the syntax elements in this sequence of bits. The header elements, the first eight bits and six syntax elements, indicate the macroblock type (P, one motion vector),

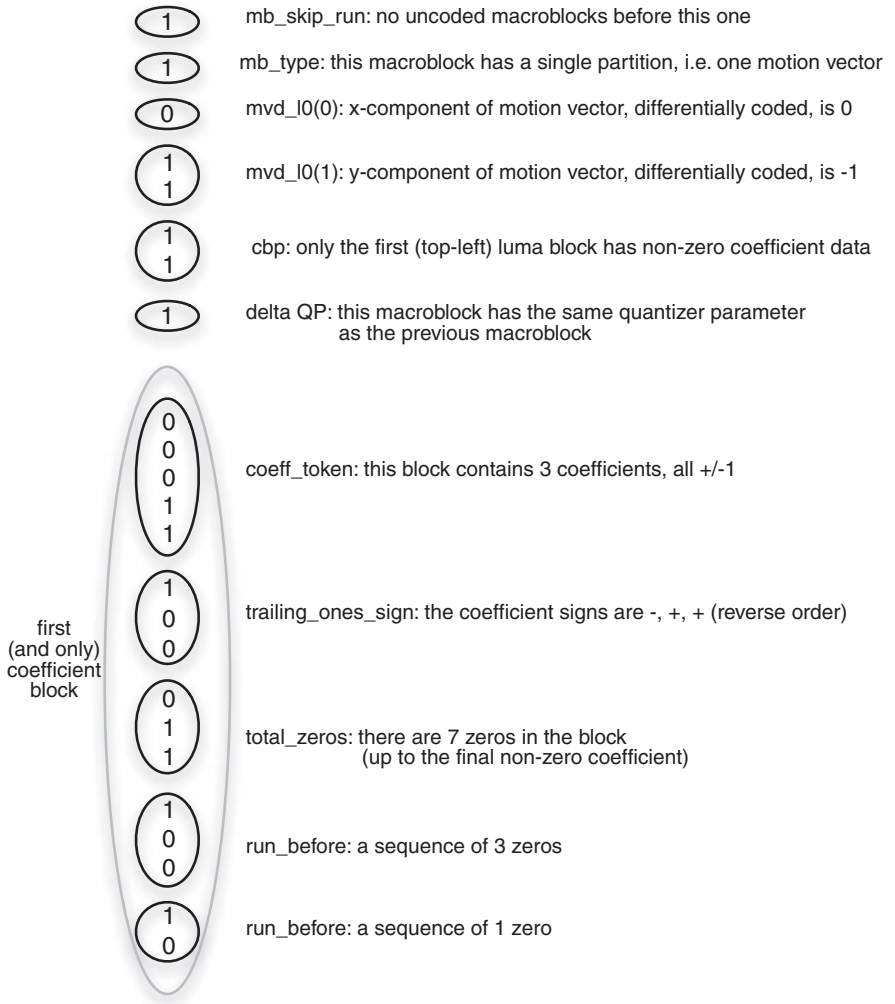


Figure 4.20 Syntax example: P-macroblock

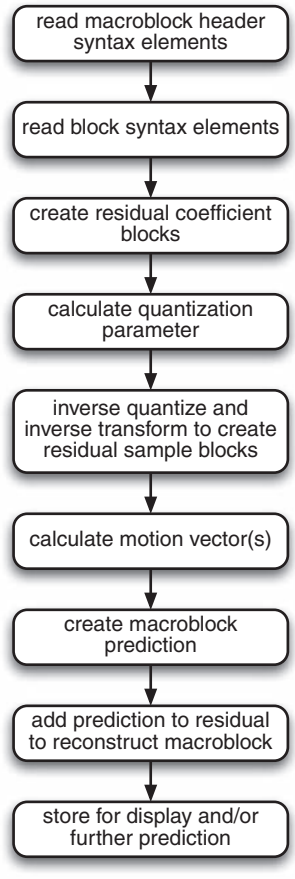


Figure 4.21 P-macroblock decoding process

the prediction parameters, in this case the x and y components of the motion vector, coded differentially from previously-received vectors, the coded block pattern which indicates that only one of the 4 × 4 blocks in the MB actually contains residual data and the quantization parameter, coded differentially from the previous macroblock. The next 16 bits represent 5 syntax elements that specify the contents of the first and only non-zero coefficient block. Decoding the macroblock proceeds as follows (Figure 4.21):

- Extract the residual luma and chroma coefficient blocks (Figure 4.22, also Chapter 7)
- Calculate the actual quantization parameter QP
- Carry out the inverse quantization and inverse transform processes to produce the residual sample blocks (Chapter 7)
- Calculate the actual motion vector
- Produce a prediction macroblock (Chapter 6)

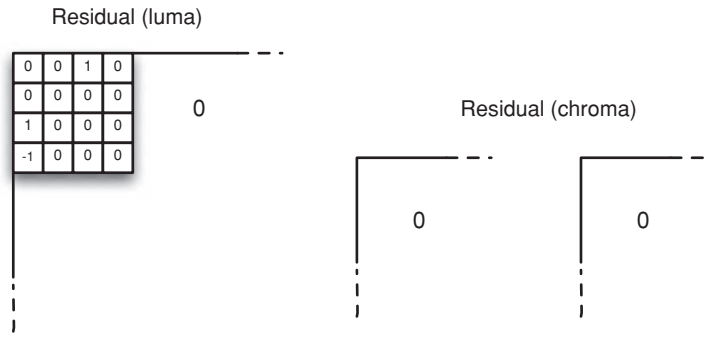


Figure 4.22 Residual luma and chroma coefficient blocks

- Add the prediction macroblock to the residual samples to reconstruct the decoded macroblock
- Store as part of the decoded frame for display and/or for further prediction (Chapters 5 and 6).

4.7 H.264 in practice

4.7.1 Performance

Perhaps the biggest advantage of H.264 over previous standards is its compression performance. Compared with standards such as MPEG-2 and MPEG-4 Visual, H.264 can deliver:

- Better image quality at the same compressed bitrate, or
- A lower compressed bitrate for the same image quality.

For example, a single-layer DVD can store a movie of around two hours' length in MPEG-2 format. Using H.264, it should be possible to store four hours or more of movie-quality video on the same disk, i.e. a lower bitrate for the same quality. Alternatively, the H.264 compression format can deliver better quality at the same bitrate compared with MPEG-2 and MPEG-4 Visual (Figure 4.23).

Figure 4.23 A video frame compressed at the same bitrate using MPEG-2 (left), MPEG-4 Visual (centre) and H.264 compression (right)

The improved compression performance of H.264 comes at the price of greater computational cost. H.264 is more sophisticated than earlier compression methods and this means that it can take significantly more processing power to compress and decompress H.264 video.

4.7.2 Applications

As well as its improved compression performance, H.264 offers greater flexibility in terms of compression options and transmission support. An H.264 encoder can select from a wide variety of compression tools, making it suitable for applications ranging from low-bitrate, low-delay mobile transmission through high definition consumer TV to professional television production. The standard provides integrated support for transmission or storage, including a packetized compressed format and features that help to minimize the effect of transmission errors.

H.264/AVC has been adopted for an increasing range of applications, including:

- High Definition DVDs (Blu-Ray)
- High Definition TV broadcasting in Europe
- Apple products including iTunes video downloads, iPod video and MacOS
- NATO and US DoD video applications
- Mobile TV broadcasting
- Many mobile video services
- Many internet video services
- Videoconferencing
- Consumer camcorders.

4.8 Summary

H.264 Advanced Video Compression is an industry standard for video coding that defines a format or syntax for compressed video and a method of decoding this syntax. It provides a set of tools or algorithms that can be used to deliver efficient, flexible and robust video compression for a wide range of applications, from low-complexity, low bitrate mobile video applications to high-definition broadcast services. The following chapters examine H.264's tools, features and performance in detail.

4.9 References

i. Recommendation ITU-T H.264 | ISO/IEC 14496-10:2009, 'Advanced Video Coding for generic audio-visual services', March 2009.
ii. ISO/IEC 14496-2:2000, 'Information technology – Generic coding of moving pictures and associated audio information: Video', 2000.

5

H.264 syntax

5.1 Introduction

An 'H.264 video' is a video sequence that is represented in a particular format, the H.264/AVC syntax. This syntax is defined in the H.264 industry standard and specifies the exact structure of an H.264-compliant video sequence, in terms of syntax elements, parameters describing different aspects of the coded sequence, and the way in which each element is represented as a binary code. The syntax is hierarchical, from the highest level, the video sequence level, down through individual coded frames or fields (access units) and subsets of access units (slices) to macroblocks and blocks. Control parameters are stored as separate syntax sections such as Parameter Sets or as part of sections such as the macroblock layer. This chapter examines the H.264/AVC syntax in detail, beginning with an overview of frame and picture structures and handling, followed by each of the main layers of the syntax hierarchy.

5.1.1 A note about syntax examples

The various syntax sections in the H.264/AVC standard are illustrated later in the chapter with extracts from coded video sequences. These were generated using the Joint Model software video codec [i] using the 'Trace' mode which is described further in Chapter 9. Each syntax example is presented in the following way, Table 5.1:

Table 5.1 Syntax examples: format

Parameter	Binary code	Symbol	Discussion
profile_idc	*1000010*	*66*	*Baseline Profile*
↑ The name of the syntax parameter, listed in the H.264/AVC standard	↑ The binary code	↑ The value decoded from this code	↑ What the value indicates.

Selected sections of the H.264 syntax are presented as syntax diagrams such as Figure 5.13.

The H.264 Advanced Video Compression Standard, Second Edition Iain E. Richardson
© 2010 John Wiley & Sons, Ltd

It should be emphasized that the syntax examples, diagrams and explanations in this chapter are for illustration only. Complete and accurate explanations of the syntax and the semantics or meaning of each syntax element are given in the H.264/AVC standard.

5.2 H.264 syntax

The hierarchical organization of the H.264/AVC syntax can be seen in Figure 5.1 [ii]. The **Network Abstraction Layer** (NAL) consists of a series of **NAL Units** (section 5.4). Sequence Parameter Sets (SPS) and Picture Parameter Sets (PPS) are NAL units that signal certain common control parameters to the decoder (section 5.5). Coded video data is communicated in **Video Coding Layer** (VCL) NAL units, known as coded **slices**. An **access unit**, a coded

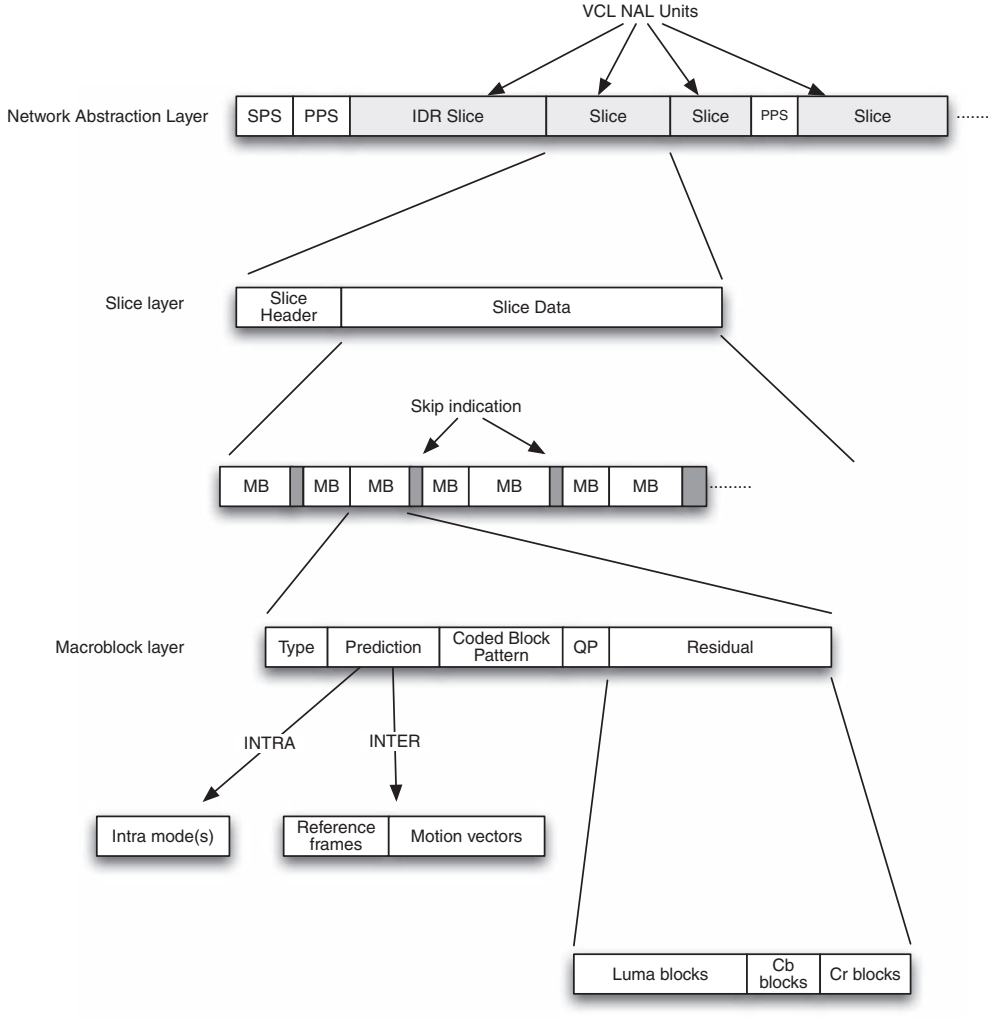


Figure 5.1 Syntax overview

frame or field, is made up of one or more slices. At the **slice layer** (section 5.6), each slice consists of a Slice Header and Slice Data. The Slice Data is a series of coded **macroblocks** (MB) and skip macroblock indicators which signal that certain macroblock positions contain no data. Each coded macroblock (section 5.7) contains the following syntax elements:

- MB type : I/intra coded, P/inter coded from one reference frame, B/inter coded from one or two reference frames.
- Prediction information : prediction mode(s) for an I macroblock, choice of reference frame(s) and motion vectors for a P or B macroblock.
- Coded Block Pattern CBP : indicates which luma and chroma blocks contain non-zero residual coefficients.
- Quantization Parameter QP, for macroblocks with CBP $\neq 0$.
- Residual data, for blocks containing non-zero residual coefficients.

A **coded video sequence** begins with an Instantaneous Decoder Refresh (**IDR**) Access Unit, made up of one or more IDR slices, a special type of Intra coded slice. Subsequent video frames or fields, described as Access Units, are coded as slices. The video sequence ends when a new IDR slice is received, signalling a new coded sequence, or when the transmission is complete.

A list of all the main sections of the syntax is given in Table 5.2. A NAL unit contains a **Raw Byte Sequence Payload (RBSP)**, a sequence of bytes containing syntax elements. H.264 syntax elements are binary codes with varying length and so a sequence of syntax elements within a NAL unit will not necessarily fit into an integral number of bytes. Zero-valued bits are added to the end of the RBSP contents in order to ensure that the length is an integral number of bytes (**RBSP trailing bits**). An RBSP syntax element can be transmitted or stored as a complete packet. Some syntax sections contain sub-sections, e.g. the Sequence Parameter Set may contain the sections *scaling_list* and *vui_parameters* and always contains the section *rbsp_trailing_bits*. Other sections, e.g. *scaling_list*, do not contain any sub-sections.

Before discussing the syntax sections in detail, it is necessary to examine the way in which H.264/AVC handles frames, fields and pictures.

5.3 Frames, fields and pictures

Chapter 2 introduced the concept of digital video and typical formats for storing and handling digital video information. An H.264 encoder converts video frames into compressed or coded picures. H.264/AVC defines a **frame** as an array of luma samples and two corresponding arrays of chroma samples. The two **fields**, top field and bottom field, that make up a frame may be sampled at the same time instant, progressive scan, or at different time instants, interlaced scan. The term **picture** refers collectively to a frame or a field. Figure 5.2 is an overview of the way pictures are handled during H.264 encoding and decoding. Frames or fields are encoded to form coded pictures, each of which is composed of one or more slices. Slices are decoded to produce decoded pictures which are stored in a Decoded Picture Buffer (DPB). Pictures in the DPB may be used for inter prediction of further coded pictures and/or output for display.

It is important to distinguish between three different orders of pictures: decoding order, the order in which pictures are decoded from the bitstream, display order, the order in which pictures are output for display, and reference order, the order in which pictures are arranged for inter prediction of other pictures (Figure 5.3).

Table 5.2 H.264 Syntax Sections

Syntax section	Description	Contains section(s)
NAL unit	Network abstraction layer unit.	Raw Byte Sequence Payload (RBSP)
Sequence Parameter Set (RBSP)	Parameters common to a video sequence.	Scaling List (optional) VUI Parameters (optional) RBSP trailing bits
Scaling List	Encoder-specified scaling matrix for inverse quantization.	None
SPS extension (RBSP)	Auxiliary picture information for alpha blending.	RBSP trailing bits
Picture Parameter Set (RBSP)	Parameters common to one or more coded pictures.	Scaling List (optional) RBSP trailing bits
Supplement Enhancement Information (RBSP)	Container for SEI message, see below.	SEI message RBSP trailing bits
Supplemental Enhancement Information Message	SEI messages contain information that may assist decoding or display but is not essential for constructing decoded video frames.	SEI payload
Access Unit Delimiter (RBSP)	Optional delimiter that indicates the slice type in the next coded picure.	RBSP trailing bits
End of Sequence (RBSP)	Optional delimiter that indicates the next slice is an IDR.	None
End of Stream (RBSP)	Optional delimiter that indicates the end of the coded bitstream.	None
Filler Data (RBSP)	Optional series of filler bytes.	RBSP trailing bits
Slice layer without partitioning (RBSP)	Coded slice that does not use data partitioning.	Slice header Slice data RBSP slice trailing bits
Slice data partition A layer (RBSP)	Partition A of a Data Partitioned Slice, see Chapter 8.	Slice header Slice data RBSP slice trailing bits
Slice data partition B layer (RBSP)	Partition B of a Data Partitioned Slice, see Chapter 8.	Slice header Slice data RBSP slice trailing bits
Slice data partition C layer (RBSP)	Partition C of a Data Partitioned Slice, see Chapter 8.	Slice header Slice data RBSP slice trailing bits
RBSP slice trailing bits	Contains RBSP trailing bits and occurs at end of coded slice or slice data partition.	RBSP trailing bits
RBSP trailing bits	Padding bits to ensure RBSP is byte aligned.	None
Slice header	Parameters common to a coded slice.	Reference picture list reordering Prediction weight table (optional) Decoded reference picture marking (optional)

Table 5.2 (*Continued*)

Syntax section	Description	Contains section(s)
Reference picture list reordering	Series of commands to change default reference picture list order(s).	None
Prediction weight table	Luma and chroma weight offsets to change effect of motion compensated prediction.	None
Decoded reference picture marking	Series of commands to mark reference pictures as long term references, etc.	None
Slice data	Contains series of coded macroblocks.	Macroblock layer
Macroblock layer	PCM samples or macroblock headers, prediction and transform coefficients.	Macroblock prediction (optional) Sub-macroblock prediction (optional) Residual data (optional)
Macroblock prediction	Intra prediction modes or reference indices and motion vectors.	None
Sub-macroblock prediction	Reference indices and motion vectors.	None
Residual data	Contains series of residual blocks, as indicated by Coded Block Pattern.	Residual block CAVLC or residual block CABAC
Residual block CAVLC	Block of transform coefficients coded using CAVLC.	None
Residual block CABAC	Block of transform coefficients coded using CABAC.	None

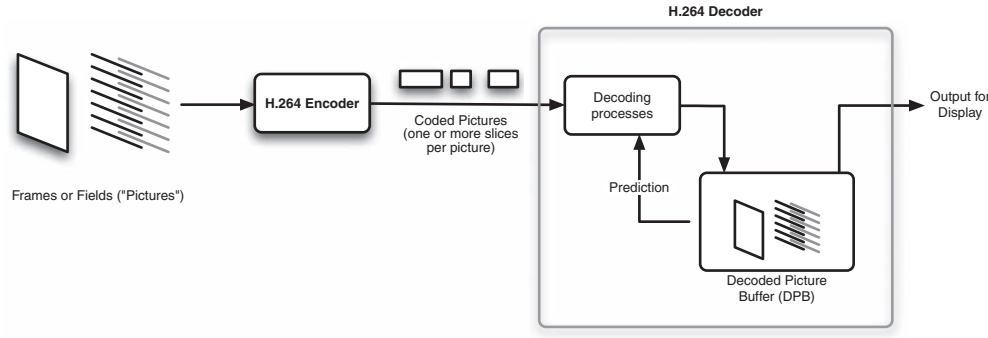


Figure 5.2 Picture handling in H.264, overview

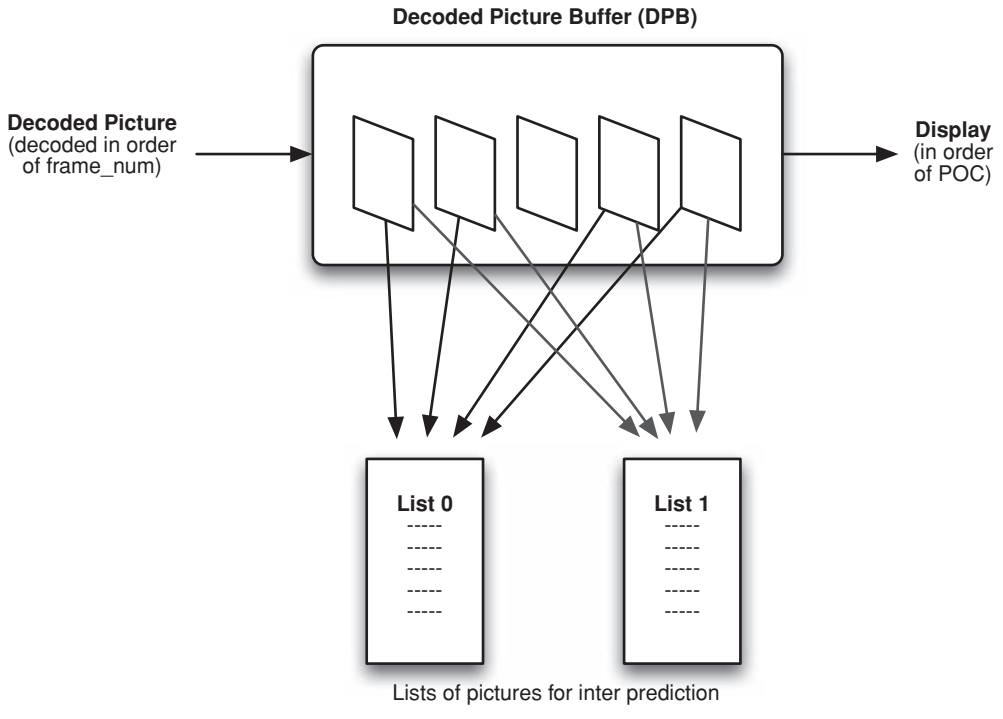


Figure 5.3 Decoded Picture Buffer and picture orders

As shown in Figure 5.3, the **decoding order** of pictures, i.e. the order in which a video decoder should process the coded pictures, is indicated by the parameter *frame_num*.

The **display order** of pictures is determined by the parameters *TopFieldOrderCount* and *BottomFieldOrderCount*, collectively described as Picture Order Count, POC.

The **reference order** of pictures is determined by one or two lists, each of which is an ordered list of all the available decoded pictures. A P slice uses a single list, list0, and a B slice uses two lists, list0 and list1, each containing available reference pictures in different orders.

5.3.1 Decoding order

The parameter *frame_num*, decoded from the slice header, determines the decoding order of pictures, coded frames or fields. Except in certain special cases, *frame_num* for each decoded picture increases by one compared with the previous **reference** frame in decoding order.

5.3.2 Display order

Display order is determined by the POC parameters *TopFieldOrderCount* and *BottomFieldOrderCount* which are derived from the slice header using one of three methods:

- Type 0: The least significant bits of POC are sent in every slice header. This allows maximum flexibility but typically requires more bits than the other methods.

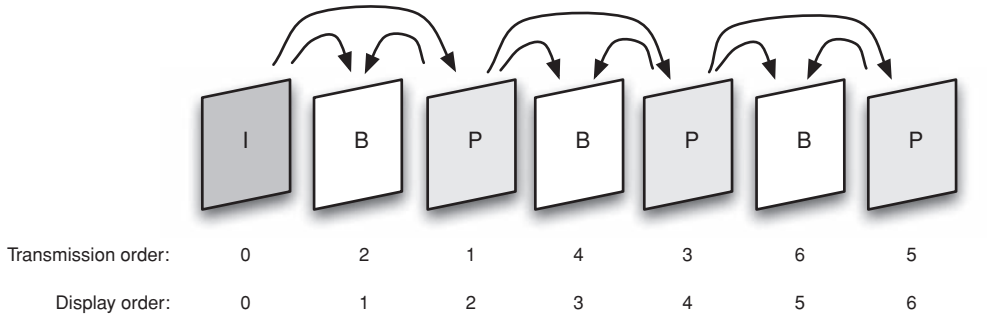


Figure 5.4 Display order: Type 0 example

- Type 1: A 'cycle' of POC increments is set up in the sequence parameter set and POC changes according to this cycle unless otherwise signalled in the slice header using a Delta offset. The cycle defines the interval between frames used for reference, plus a POC offset to frames not used for reference.
- Type 2: POC is derived directly from *frame_num* and display order is the same as decoding order.

Display Order Example, Type 0:

Frame pictures, one frame per slice, display order IBPBPBPBPB. . . (Figure 5.4).
In this example, the B slices, bipredicted slices in which each macroblock or partition can be predicted from up to two reference pictures, are not used for reference prediction of any other pictures. POC increments by two for every complete frame, i.e. every two fields. Note that *frame_num* increments **after** each reference picture is transmitted. In this example, only the I and P pictures are reference pictures. Increments in *frame_num* are indicated in bold type.

Slice	Type	Used for reference	*frame_num*	POC LSBs	Display order
1st	I	Yes	**0**	0	0
2nd	P	Yes	**1**	4	2
3rd	B	No	**2**	2	1
4th	P	Yes	2	8	4
5th	B	No	**3**	6	3
6th	P	Yes	3	12	6
7th	B	No	**4**	10	5
8th	P	Yes	4	16	8
. . .	. . .				

Display Order Example, Type 1:

Frame pictures, one frame per slice, display order: IBBPBBPBBPB. . . (Figure 5.4)

In this example, the B slices are not used for reference. The POC cycle consists of one reference frame, with an offset from the previous reference frame to the next reference frame of +6 and an offset from the previous reference frame to the next non-reference frame of −4.

Slice	Type	Used for reference	*frame_num*	Delta POC	Offset	POC	Display order
1st	I	Yes	**0**	0	0	0	0
2nd	P	Yes	**1**	0	+6	6	3
3rd	B	No	**2**	0	−4	2	1
4th	B	No	2	2	−4+2	4	2
5th	P	Yes	2	0	+6	12	6
6th	B	No	**3**	0	−4	8	4
7th	B	No	3	2	−4+2	10	5
8th	P	Yes	3	0	+6	18	9
...	...						

The 4th and 7th slices have a delta POC, signalled in the slice header, of +2. The POC for these slices is calculated as: POC = expected POC + 2.

5.3.3 Reference picture lists

A picture that is coded and available for reference is stored in the Decoded Picture Buffer (DPB) and marked as one of the following:

(a) a short term reference picture, indexed according to *frame_num* or POC or,
(b) a long term reference picture, indexed according to *LongTermPicNum*, a reference number derived from the parameter *LongTermFrameIdx* which is assigned when a picture is marked as a long term reference picture.

Short term reference pictures may be assigned a *LongTermFrameIdx*, i.e. changed to a long term reference picture, at a later time.

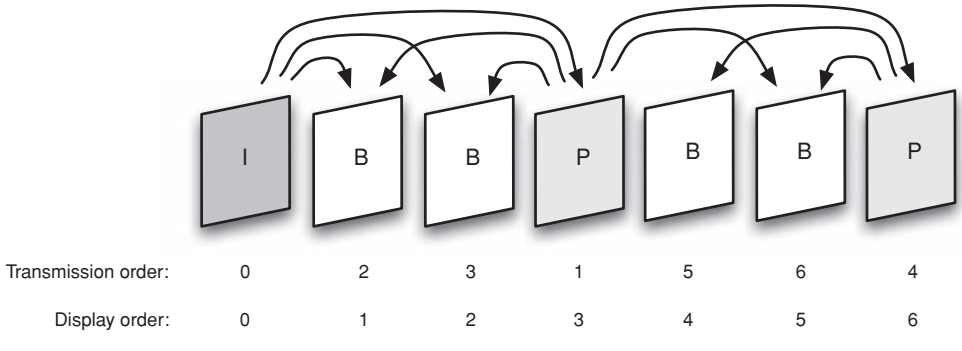


Figure 5.5 Display order: Type 1 example

Short term reference pictures are removed from the DPB (a) by an explicit command in the bitstream or (b) when the DPB is full and an automatic mode of DPB handling is in use, in which case the oldest short term picture is removed. Long term pictures are removed by an explicit command in the bitstream.

5.3.3.1 Default reference picture list order

Reference pictures are ordered in one or two lists prior to encoding or decoding a slice. A P slice uses a single list of reference pictures, list0 and a B slice uses two lists, list0 and list1. In each list, short term reference pictures are listed first followed by long term reference pictures, in increasing order of *LongTermPicNum*. The default short term reference picture order depends on **decoding order** when the current slice is a P slice and on **display order** when the current slice is a B slice. The list orders are important, since indices to reference pictures earlier in the list require fewer bits to signal. Hence the default orders are organized so that reference pictures temporally 'nearer' to the current picture occur earlier in the list, since these are likely to contain the best prediction match for the current picture.

List0 (P slice) : The default order is in decreasing order of *PicNum*. *PicNum* is a version of *frame_num*, 'wrapped around' modulo *MaxFrameNum*.

List0 (B slice) : The default order is (1) in decreasing order of POC, for pictures with POC earlier than current picture, then (2) in increasing order of POC, for pictures with POC later than the current picture.

List1 (B slice) : The default order is (1) in increasing order of POC, for pictures with POC later than the current picture, then (2) in decreasing order of POC, for pictures with POC earlier than current picture.

Reference Picture Order Example: P slices (list0)

The reference picture list is initially empty. The current *frame_num* is 150 and the maximum size of the DPB is 5 frames. Italics indicate a *LongTermPicNum*. Notice that by default, the first entry in list0 is the most recently coded frame. This is likely to be the 'best' prediction reference for the majority of macroblocks.

Operation	list0(0)	list0(1)	list0(2)	list0(3)	list0(4)
Initial state	-	-	-	-	-
Encode frame 150	150	-	-	-	-
Encode 151	151	150	-	-	-
Encode 152	152	151	150	-	-
Encode 153	153	152	151	150	-
Encode 154	154	153	152	151	150
Encode 155	155	154	153	152	151
Assign 154 to LongTermPicNum *3*	155	153	152	151	*3*
Encode 156 and mark it as LongTermPicNum *1*	155	153	152	*1*	*3*
Encode 157	157	155	153	*1*	*3*
. . ...					

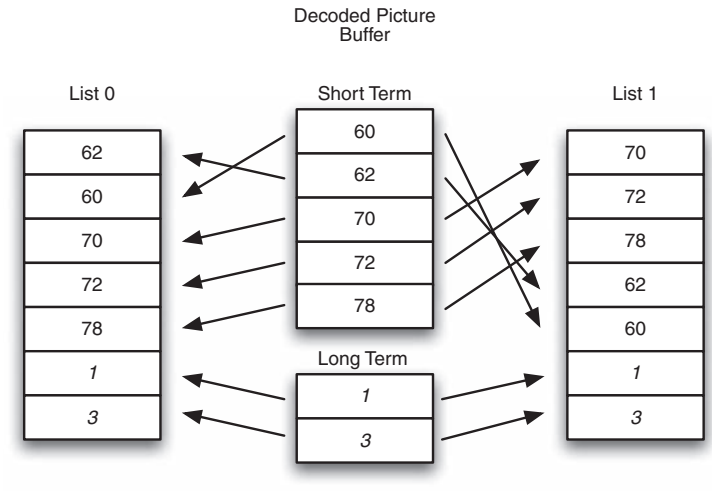


Figure 5.6 List 0 and List 1 ordering: example

Reference Picture Order Example: B slice (list0 and list1)

The DPB contains short term reference pictures with POC 60, 62, 70, 72 and 78 and long term reference pictures with LongTermFrameIdx *1* and *3*. The current POC is 68. The default order of list0 is:

62, 60, 70, 72, 78, *1, 3*

The default order of list1 is:

70, 72, 78, 62, 60, *1, 3*

Figure 5.6 shows how the two lists are created from the sequence of pictures in the DPB. List0 defaults to prediction from 'past' pictures in display order whilst List1 defaults to 'future' pictures in display order. This is shown graphically in Figure 5.7. Note that the first entry in

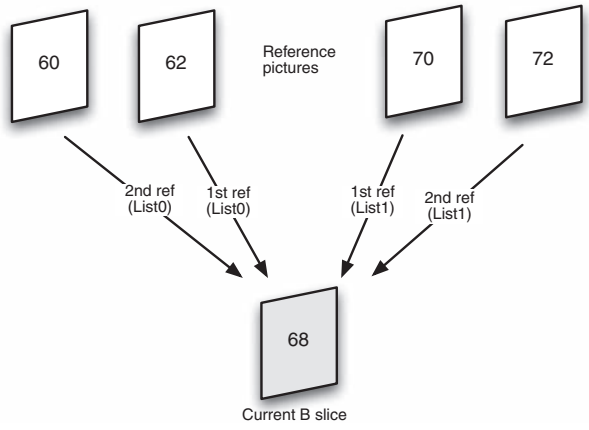


Figure 5.7 Default reference pictures: example

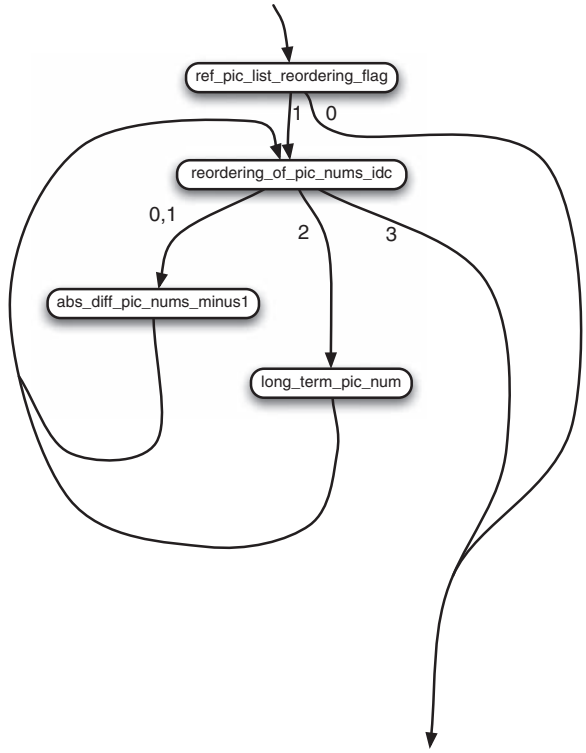


Figure 5.8 Reference picture re-ordering syntax, simplified overview

list0 is the most recent **past** picture (62) and the first entry in list1 is the nearest **future** picture (70). These are likely to be the best 'past' and 'future' prediction references respectively.

5.3.3.2 Changing the reference picture list order

The default order of list0, and of list1 in the case of a B slice, may be modified by explicit commands in the slice header. These commands are initiated by the syntax element *ref_pic_list_reordering_flag* and change the list order **for the current slice only**. This may be used, for example, to enable an encoder to place a particularly useful reference frame earlier in the list than its usual position, since earlier reference indices cost fewer bits to signal.

The reference picture list re-ordering process is illustrated in Figure 5.8 and proceeds as follows for list 0. This is a simplified overview.

- Initialize a pointer (refIdxL0) to point to the first reference picture index in the list (0)
- While *reordering_of_pic_nums_idc* $\neq 3$
  - Select a reference picture, either short term, indicated by *abs_diff_pic_num_minus1*, or long term, indicated by *long_term_pic_num*
  - Move all other pictures from position refIdxL0 onwards in the list to one position later
  - Put this picture in the list at the position indicated by refIdxL0
  - Increment pointer refIdxL0.

Table 5.3 Selecting a reference picture to re-map

reordering_of_pic_nums_idc	remappedPicNum is calculated as:
0	predictedPicNum – (*abs_diff_pic_num_minus_1* +1)
1	predictedPicNum + (*abs_diff_pic_num_minus_1* +1)
2	*long_term_pic_num*
3	(Exit the re-ordering process)

The value *abs_diff_pic_num_minus1*+1 signals a positive or negative offset from a **predicted** reference picture number. For the first reordering or remapping instruction, the predicted picture number predictedPicNum is the current picture number. For subsequent reordering instructions, the predicted picture number is the picture number of the last remapped picture. The picture to be remapped or moved, remappedPicNum, is selected according to the following rules shown in Table 5.3.

Reference Picture Re-Ordering Example:
The current slice is a P slice. The DPB contains five reference frames, indexed in list0. Three of these are short-term frames with PicNum equal to 153, 155, and 157, respectively. The other two are long-term frames with LongTermPicNum equal to *1* and *3*, respectively. The current *frame_num* is 158. The default reference list order (list0) is as follows (Table 5.4).

Table 5.4 Initial reference picture list

Index	0	1	2	3	4
Picture	157	155	153	*1*	*3*

The following series of four reference picture reordering commands are received.
Initial predictedPicNum = 158, the current picture; initial refIdxL0 = 0

1. *reordering_of_pic_nums_idc = 0, abs_diff_pic_num_minus_1 = 4*
   remappedPicNum = 158 – 5 = 153
   Put picture 153 at position refIdxL0=0 (the start of the list).
   New list: **153**, 157, 155, *1*, *3*
   New predictedPicNum = 153; new refIdxL0 = 1.
2. *reordering_of_pic_nums_idc = 1, abs_diff_pic_num_minus_1 = 1*
   remappedPicNum = 153 + 2 = 155
   Put picture 155 at position refIdxL0=1, second in the list.
   New list: 153, **155**, 157, *1, 3*
   New predictedPicNum = 155; new refIdxL0 = 2.
3. *reordering_of_pic_nums_idc = 2, long_term_pic_num* = 3
   remappedPicNum= *3*
   Put long term picture *3* at position refIdxL0=2, third in the list.
   New list: 153, 155, **3**, 157, *1*
4. *reordering_of_pic_nums_idc* = 3
   End of reordering process.

The final list0 order is as follows (Table 5.5).

Table 5.5 Final reference picture list

Index	0	1	2	3	4
Picture	153	155	*3*	157	*1*

5.3.4 Frame and field coding

A video sequence may be coded in Frame mode only, with no special coding of fields or interlaced video, or in Frame/Field mode, using special coding tools for interlaced video, signalled by the syntax element *frame_mbs_only_flag* in the Sequence Parameter Set (section 5.5). If this flag is set to zero, special coding of fields or interlaced video is enabled.

5.3.4.1 Coding pictures in frame or field mode

If frame/field coding is enabled, each frame of video, a pair of fields, may be coded as a complete frame, i.e. one picture = one frame, or as two separate fields, i.e. one picture = one field. The syntax element *field_pic_flag* is signalled in each Slice Header (section 5.6.2) and indicates whether the current coded picture is a frame or a field. This gives rise to the following two cases:

1. Picture is coded as a complete frame: The complete frame, which happens to include two fields, is coded as one or more slices, each containing an integral number of macroblocks. Reference picture list(s), used for motion compensated prediction of P- or B-macroblocks, are constructed from previously coded **frames**, i.e. each reference picture is a complete coded frame. Block coefficients are scanned in a zig-zag order (Chapter 7). Macroblocks are optionally coded as macroblock pairs in Macroblock Adaptive Frame Field Mode.
2. Picture contains one field: The current top or bottom field is coded as one or more slices, each containing a number of macroblocks. Reference picture list(s) are constructed from previously coded **fields**, i.e. each reference picture is a coded field. Each field of a stored reference frame is identified as a separate reference picture with a unique index[1]. An inter-coded macroblock is predicted from region(s) in previously coded field(s). Block coefficients are coded in a modified or field scan order (Chapter 7).

The coded frames in the Decoded Picture Buffer (DPB) may therefore be accessed as complete frames, containing top + bottom field, or as individual fields, depending on whether the current picture is coded in frame or field mode.

An encoder may choose to switch adaptively between frame and field coding during a sequence, for example to improve the compression performance for interlaced video or for progressive video that was converted from an interlaced source. For an interlaced video

[1] When decoding a field, there are effectively twice as many reference pictures available as there would be when decoding a frame at the same position.

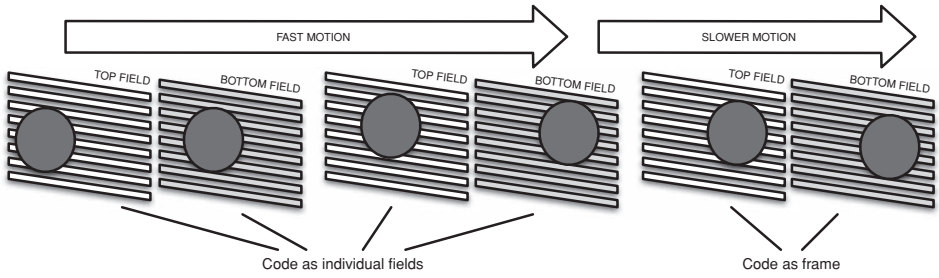


Figure 5.9 Picture Adaptive Frame Field Coding example

sequence, field coding tends to be more efficient when there is significant motion in the scene which tends to give larger changes between successive fields, whereas frame coding tends to be more efficient in more static areas of the scene where there is less change between successive fields. Switching between frame and field coding at the picture level is known as Picture Adaptive Frame Field Coding (PAFF).

Example:

Figure 5.9 shows a sequence of six fields from an interlaced sequence or from a progressive sequence that was converted from interlaced content. The first four contain rapid motion, a fast-moving ball, and so the encoder chooses to code each field as an individual picture using field coding mode. Each field is coded as a unit; reference picture list(s) consist of previously-coded fields; block scanning uses the modified field scan described in Chapter 7. The last two fields contain slower motion and the encoder chooses to code these as a complete frame. The combined frame is coded as a single picture; reference picture list(s) consist of previously-coded frames; block scanning uses the zig-zag progressive scan described in Chapter 7.

5.3.4.2 Coding macroblocks in frame or field mode (MBAFF)

Macroblock Adaptive Frame Field Coding (MBAFF) [iii] is a coding mode that is enabled in the Sequence Parameter Set (section 5.5) and is available thereafter in any slice coded in Frame mode. MBAFF makes it possible to adaptively switch between frame and field coding within a single frame, for example to maximise coding efficiency. Macroblocks are handled in pairs consisting of two vertically adjacent macroblocks (Figure 5.10).

For each macroblock pair, *mb_field_decoding_flag*, signaled in the header of the upper macroblock, indicates whether the pair is coded as a frame macroblock pair or a field macroblock pair (Figure 5.11). If it is a **frame MB pair**, then the top and bottom macroblocks are coded separately as frame macroblocks. Blocks are scanned using zig-zag scan (Chapter 7). If it is a **field MB pair**, alternate lines of the MB pair are mapped to top and bottom field macroblocks as shown in the figure. Each field macroblock is predicted from the corresponding field in reference picture(s) (Chapter 6). Blocks in a field MB are scanned using field scan (Chapter 7).

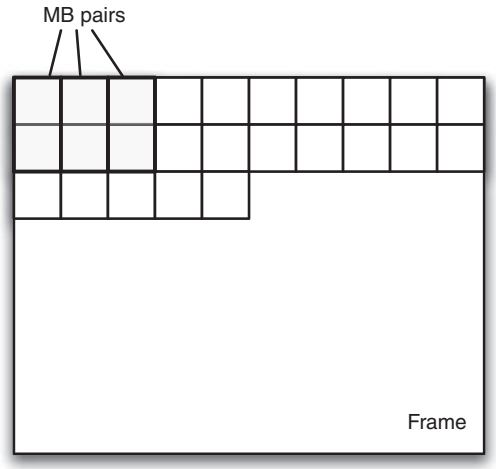


Figure 5.10 Frame with macroblock pairs

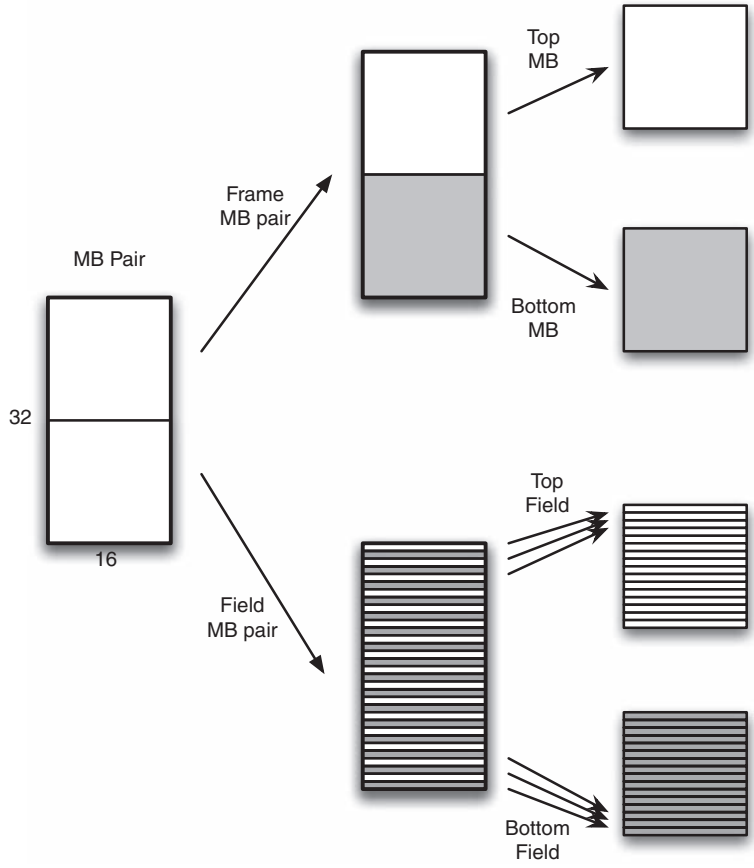


Figure 5.11 MB pair coded using Macroblock Adaptive Frame Field Coding

5.4 NAL unit

Coded H.264 data is stored or transmitted as a series of packets known as Network Abstraction Layer Units, NAL Units or NALUs. Each NAL Unit consists of a 1-byte NALU header followed by a byte stream containing control information or coded video data. The header indicates the NALU type, some of which are listed in Table 5.6, and the 'importance' of the NALU. Parameter Sets and slices that are used for reference, i.e. used to predict further frames, are considered important or high priority, since their loss could make it difficult to decode subsequent coded slices. Non-reference slices are considered to be less important to the decoder, since their loss will not affect any further decoding. This information can optionally be used to prioritise certain NALUs during transmission (Chapter 8).

Coded slices are described as Video Coding Layer (VCL) NAL units. A coded sequence begins with an Instantaneous Decoder Refresh (IDR) Access Unit, made up of one or more IDR slices, each of which is an Intra coded slice. This is followed by the default slice type, i.e. a non-IDR coded slice, and/or by Data Partitioned slices. Data Partitioned slices carry different components of coded video data in separate NAL units, which may be useful in error-prone transport situations. Non-VCL NAL units include Parameter Sets, Supplemental Enhancement Information parameters that may be useful for decoding and displaying video data, but are not essential for correct decoding, and delimiters that define boundaries between coded sections.

Each NALU may be transported in one or more network packets, streamed sequentially as a byte stream or encapsulated in a media file (Chapter 8).

Table 5.6 Selected NAL unit types

NAL Unit	Description	VCL
Coded slice, non-IDR	A typical coded slice, see section 5.6	Yes
Coded slice data partition A	Part of a data partitioned slice, see Chapter 8	Yes
Coded slice data partition B	Part of a data partitioned slice, see Chapter 8	Yes
Coded slice data partition C	Part of a data partitioned slice, see Chapter 8	Yes
Coded slice, IDR	Part of the Access Unit at the start of a coded video sequence	Yes
SEI	Supplemental Enhancement Information, see Chapter 8	No
SPS	Sequence Parameter Set, one per sequence, see section 5.5	No
PPS	Picture Parameter Set, see section 5.5	No
Access unit delimiter	Indicates type of the slices in the next coded picture (optional)	No
End of sequence	Indicates that the next access unit is IDR, i.e. the start of a new coded video sequence (optional)	No
End of stream	Indicates the end of the coded bitstream (optional)	No
Filler	Filler bytes (optional), e.g. to prevent buffer under-run, see Chapter 8.	No
.....		

5.5 Parameter Sets

Parameter Sets are NAL units that carry decoding parameters common to a number of coded slices. Sending these parameters independently of coded slices can improve efficiency, since common parameters need only be sent once. Parameter Sets are essential to the correct decoding of a sequence. In a lossy transmission scenario, where bits or packets may be lost or corrupted during transmission, Parameter Sets may be sent with a higher quality of service using e.g. Forward Error Correction or a priority mechanism.

A Sequence Parameter Set (SPS) contains parameters common to an entire video sequence or programme, such as the Profile and Level (Chapter 8), the size of a video frame and certain decoder constraints such as the maximum number of reference frames (section 5.3.3). Table 5.7 shows an example of a SPS for a Baseline Profile, QCIF coded sequence. Note that each SPS has a unique identifier, *seq_parameter_set_id*, which is 0 in this case.

A Picture Parameter Set (PPS) contains common parameters that may apply to a sequence or subset of coded frames, such as entropy coding type, number of active reference pictures and initialization parameters. See Table 5.8 for an example. The PPS has its own identifier, *pic_parameter_set_id*, and 'points' to a SPS identifier. In this example, the PPS 'inherits' the parameters of SPS 0. Each coded slice 'points' to a PPS and inherits its parameters. In this way, there is considerable flexibility to set up common coding parameters via SPS and PPS NAL units and then to refer to these in subsequent coded slices.

Table 5.7 Sequence Parameter Set example

Parameter	Binary code	Symbol	Discussion
profile_idc	1000010	66	Baseline Profile
constrained_set0_flag	0	0	Bistream might not obey all the constraints of the Baseline Profile
constrained_set1_flag	0	0	As above, Main Profile
constrained_set2_flag	0	0	As above, Extended Profile
constrained_set3_flag	0	0	Used to specify the special case of Level 1b
reserved_zero_4bits	0	0	Not used
level_idc	11110	30	Level 3
seq_parameter_set_id	1	0	Sequence Parameter Set 0
log2_max_frame_num_minus4	1	0	frame_num will not exceed 16.
pic_order_cnt_type	1	0	Default POC
log2_max_pic_order_cnt_lsb_minus4	1	0	LSB of POC will not exceed 16.
num_ref_frames	1011	10	Up to 10 reference frames.
gaps_in_frame_num_value_allowed_flag	0	0	No gaps in frame_num.
pic_width_in_mbs_minus1	1011	10	11 macroblocks wide = QCIF
pic_height_in_map_units_minus1	1001	8	9 MBs high = QCIF
frame_mbs_only_flag	1	1	No field slices or field MBs
direct_8 × 8_inference_flag	1	1	Specifies how certain B macroblock motion vectors are derived
frame_cropping_flag	0	0	Frames are not cropped
vui_parameters_present_flag	0	0	VUI parameters not present

Table 5.8 Picture Parameter Set example

Parameter	Binary code	Symbol	Discussion
pic_parameter_set_id	1	0	Picture Parameter Set 0
seq_parameter_set_id	1	0	Use SPS 0
entropy_coding_mode_flag	0	0	CAVLC entropy coding
pic_order_present_flag	0	0	POC not present
num_slice_groups_minus1	1	0	One slice group
num_ref_idx_l0_active_minus1	1010	9	10 reference pictures in list0
num_ref_idx_l1_active_minus1	1010	9	10 reference pictures in list1
weighted_pred_flag	0	0	Weighted prediction not used
weighted_bipred_idc	0	0	Weighted biprediction not used
pic_init_qp_minus26	1	0	Initial QP (luma) = 26
pic_init_qs_minus26	1	0	Initial SI/SP QP=26
chroma_qp_index_offset	1	0	No chroma QP offset
deblocking_filter_control_present_flag	0	0	Use default filter parameters
constrained_intra_pred_flag	0	0	Intra prediction is not constrained
redundant_pic_cnt_present_flag	0	0	Redundant picture count parameter is not used

Activation of Parameter Sets

A Parameter Set is 'inactive' until it is activated, i.e. its parameters are not used by the decoder. A PPS, previously transmitted to the decoder, is activated when it is referred to in a slice header and remains active until a different PPS is activated. A SPS is activated when a PPS that refers to it, e.g. the PPS in Table 5.8, is activated. A single SPS remains active for an entire coded video sequence, which must start with an IDR Access Unit, hence a SPS is effectively activated by an IDR slice. Each coded slice in a video sequence may refer to one of a number of Picture Parameter Sets.

Using parameter sets: Example

SPS0 is sent at the start of a sequence, followed by PPS0 and PPS1, both of which 'inherit' SPS0 (Figure 5.12). PPS0 is activated by IDR slice 0, which means that SPS0 becomes active at the same time. Slices 1 and 2 use the parameters of PPS0 and SPS0. PPS1 is activated by slice 3, making PPS0 inactive, hence slice 3 uses the parameters of PPS1 and SPS0; PPS0 is activated again by slice 4, making PPS1 inactive.

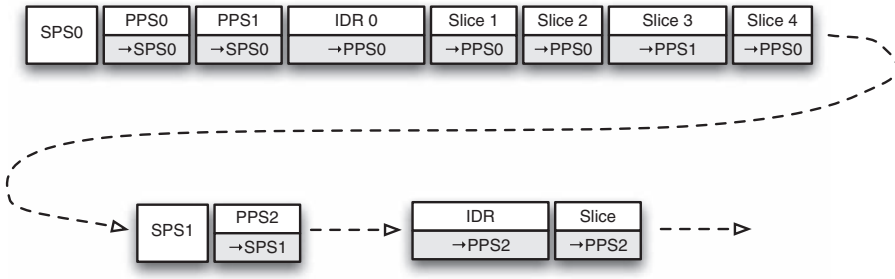


Figure 5.12 Example: Sequence and Picture Parameter Sets

At a later point, SPS1 and PPS2 are sent; PPS2 inherits SPS1. Both SPS1 and PPS2 are inactive until a new IDR slice is sent, activating both SPS1 and PPS2 and signalling the start of a new coded video sequence. Subsequent slices use the parameters of SPS1 and PPS2.

5.6 Slice layer

5.6.1 Slice types

Each coded picture, a coded frame or field, is composed of one or more slices, each containing a **slice header** followed an integral number of macroblocks. The number of macroblocks in a slice need not be constant. There are minimal data inter-dependencies between coded slices, which can help to limit the propagation of errors between slices. Possible scenarios for choosing slice sizes include:

- One slice per coded picture, common practice in many H.264 encoding applications.
- N slices per picture, each containing M macroblocks, N and M integer. The number of bytes in each coded slice will tend to vary depending on the amount of motion and detail in the picture area.
- N slices per picture, containing a varying number of macroblocks, chosen to keep the number of bytes per slice roughly constant. This may be useful if, for example, each slice is mapped to a fixed-size network packet.

The available slice types are listed in Table 5.9, together with the macroblock types that may be present in each slice. Note that a B slice (for example) may contain B, P and/or I macroblock types and an encoder may choose between these types depending on the content of the sequence. Macroblock types are explained further in section 5.7.1.

5.6.2 Slice header

The slice header conveys information common to all macroblocks in the slice, such as the slice type which determines which macroblock types are allowed, the frame number that the slice corresponds to, reference picture settings and default quantization parameter (QP). Table 5.10

Table 5.9 Slice types in H.264

Slice type	Contains macroblock types	Notes
I (including IDR)	I only	Intra prediction only.
P	I and/or P	Intra prediction (I) and/or prediction from one reference per macroblock partition (P).
B	I, P and/or B	Intra prediction (I), prediction from one reference (P) or biprediction, i.e. prediction from two references (B)
SP	P and/or I	Switching P slice, see Chapter 8
SI	SI	Switching I slice, see Chapter 8

Table 5.10 Slice Header, IDR/Intra, Frame 0

Parameter	Binary code	Symbol	Discussion
first_mb_in_slice	1	0	First MB is at position 0, the top-left position in the slice.
slice_type	1000	7	I slice, contains only I MBs
pic_parameter_set_id	1	0	Use PPS 0
frame_num	0	0	Slice is in frame 0
idr_pic_id	1	0	IDR #0 : only present in IDR picture
pic_order_cnt_lsb	0	0	Picture order count = 0
no_output_of_prior_pics_flag	0	0	Not used
long_term_reference_flag	0	0	Not used
slice_qp_delta	1000	4	QP = initial QP + 4 = 30

shows an example of a slice header from an I slice with frame_mbs_only_flag set, i.e. no field coding. This is the first I slice and is therefore an IDR slice. The default QP is set to the initial sequence value (26) plus 4.

The header of a P slice is shown in Table 5.11. This is frame 1; the picture order count (POC) is 2. Note that POC increases by 2 for every complete frame, see section 5.3.2. There is one active reference picture which happens to be frame 0.

5.6.3 Slice data

The slice data section consists of a series of macroblocks that make up a slice. A macroblock containing no data, a skip macroblock, is a very common occurrence in many coded sequences. It is signalled by the parameters mb_skip_run, a count of a sequence of skipped macroblocks, used with CAVLC entropy coding mode, or mb_skip_flag, indicating a single skipped macroblock, used with CABAC entropy coding mode. A skip macroblock can only

Table 5.11 Slice Header, Inter, Frame 1

Parameter	Binary code	Symbol	Discussion
first_mb_in_slice	1	0	First MB at position 0
slice_type	110	5	P slice, can contain I or P MBs
pic_parameter_set_id	1	0	Use PPS 0
frame_num	1	1	Frame 1
pic_order_cnt_lsb	10	2	Picture 2
num_ref_idx_active_override_flag	1	1	Default number of reference pictures overridden by following parameter
num_ref_idx_l0_active_minus1	1	0	One reference picture in List 0
ref_pic_list_reordering_flag_l0	0	0	No re-ordering of reference pictures
adaptive_ref_pic_buffering_flag	0	0	Reference pictures handled as 'first in / first out' (default)
slice_qp_delta	1000	4	QP = 26 + 4 = 30

occur in a P, SP or B slice. A complete slice data section consists of coded and skipped macroblocks that comprise a single coded slice (Figure 5.1).

5.7 Macroblock layer

5.7.1 Overview

The Macroblock Layer contains all the syntax elements necessary to decode a single macroblock (Figure 5.13).

mb_type indicates the macroblock coding type, I, SI, P or B, and further information about macroblock prediction and coding. An I macroblock (Intra) can occur in any slice type and is coded without reference to any other slices. An SIntra MB (SI) occurs only in switching or SI

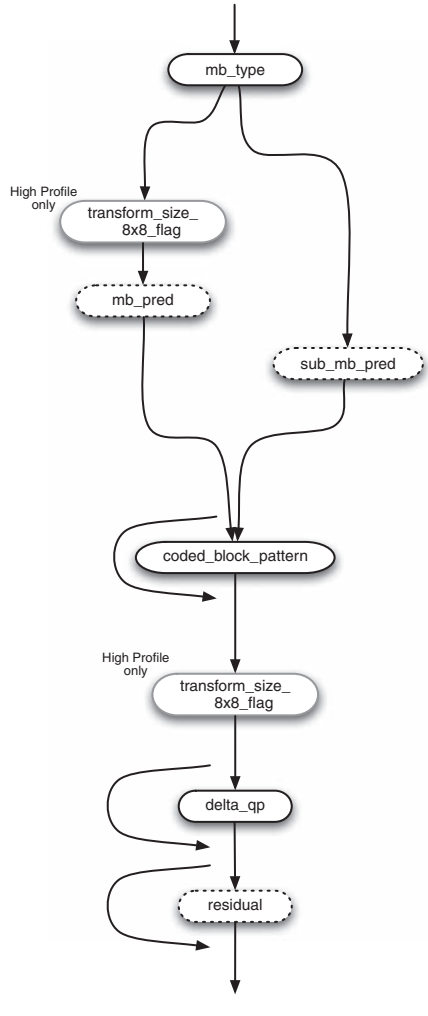


Figure 5.13 Macroblock layer syntax overview

Table 5.12 Macroblock types

Macroblock coding type	Further information signalled in mb_type codeword
I	4×4 : indicates 4×4 Intra prediction mode.
	8×8 : indicates 8×8 Intra prediction mode.
	16×16 : indicates 16×16 Intra prediction mode; luma and chroma coded block pattern is also indicated by mb_type.
	PCM: indicates that the special PCM mode is used (section 5.7.2).
SI	None.
P	16×16, 16×8, 8×16: indicates partition size.
	8×8: indicates 8×8 partition size, a further syntax element sub_mb_type and optionally a reference index are also transmitted.
	P_Skip (inferred): indicates Skip mode, no further data transmitted.
B	16×16: indicates 16×16 partition size. Prediction mode may be Direct, no motion vectors or references transmitted, Pred, prediction from one reference picture in list 0 or list 1, or BiPred, prediction from two references, one from list 0 and one from list 1.
	16×8: indicates 16×8 partition size plus prediction modes of each partition, either Pred from L0, Pred from L1 or BiPred.
	8×16: indicates 8×16 partition size plus prediction modes of each partition, Pred from L0 or L1, or BiPred.
	8×8: indicates 8×8 partition size. A further syntax element sub_mb_type is sent for each 8×8 partition.
	B_Skip (inferred): indicates Skip mode, no further data transmitted.

slices (Chapter 8). A P macroblock (Predicted) can occur in P or SP slices and is inter-coded with one prediction reference. A B macroblock (BiPredicted) can occur in B slices and is inter-coded with one or two prediction references. Table 5.12 summarizes the information signalled by the codeword mb_type. Note that mb_type is not actually transmitted for skip MBs in P or B slices – these macroblock types are inferred whenever mb_skip is signalled in the slice data layer.

transform_size_8 × 8_flag is only present in High Profile bitstreams. It occurs in one of two places (Figure 5.13) depending on the macroblock type and does not occur in an Intra 16×16 macroblock. This flag indicates that an optional 8×8 integer transform is used for luma blocks, instead of the default 4×4 integer transform (Chapter 7).

mb_pred, for all macroblock types except P/B with 8×8 partition size, or **sub_mb_pred**, for P/B macroblocks with 8×8 partition size, indicates the type of intra or inter prediction used for the macroblock (section 5.7.3 and Chapter 6).

coded_block_pattern is a separate syntax element for macroblocks other than Intra_16×16_types. It can take values between 0 and 47 and is constructed as follows:

(i) The four LSBs of coded_block_pattern, $b_3b_2b_1b_0$, indicate whether there are one or more non-zero transform coefficients within each of the four 8×8 luma blocks. A 1 = non-zero coefficient(s) present, 0 = no coefficients present.

Table 5.13 coded_block_pattern examples.

Pattern : ✓ = coefficients MAY BE present, 0 = no coefficients	Coded_block_pattern
Luma 8x8 blocks: ✓ ✓ 0 ✓; Chroma DC: ✓; Chroma AC: 0 0	$\mathbf{01}1011_2 = 27_{10}$
Luma 8x8 blocks: 0 0 0 ✓; Chroma DC: 0; Chroma AC: 0 0	$\mathbf{00}1000_2 = 8_{10}$
Luma 8x8 blocks: ✓ ✓ ✓ ✓; Chroma DC: ✓; Chroma AC: ✓ ✓	$\mathbf{10}1111_2 = 47_{10}$

(ii) The two MSBs, b_5b_4, can take the values 00_2, indicating no chroma coefficients present, 01_2, indicating chroma DC coefficients are present, no chroma AC coefficients, or 10_2, indicating chroma DC coefficients may be present, chroma AC coefficients are present.

Table 5.13 shows three examples; a tick indicates that coefficients may be present in the block, a 0 indicates that no non-zero coefficients are present.

delta_qp indicates a change in quantization parameter (QP), either positive or negative, from its previous value. If there is no change in QP from the previous value, delta_qp = 0.

If residual coefficient data is present, i.e. if there are non-zero coefficients in the macroblock, as indicated by a non-zero CBP, **residual_data** is then transmitted (section 5.7.4).

5.7.2 The Intra PCM mode

The Intra PCM (Pulse Code Modulation) mode is an optional mode of coding a macroblock, signalled by mb_type I_PCM. If I_PCM mode is chosen, the usual prediction, transform and

coding processes are bypassed. Instead, each luma and chroma sample is transmitted directly as follows:

1. Send zero bits until the current position in the bitstream is byte aligned.
2. Send every luma sample, 256 samples in total, as an individual BitDepth_Y-bit value.
3. Send every chroma sample, 64 samples for 4:2:0 format, as an individual BitDepth_c-bit value.

BitDepth_Y and BitDepth_C are the number of bits used to represent an uncompressed luma or chroma sample respectively. Each BitDepth parameter defaults to 8, i.e. 1 byte per sample, but higher bit depths may be set in the SPS for certain Profiles (Chapter 8).

Clearly, I_PCM mode does not provide compression, as the 'raw' luma and chroma samples are simply inserted into the H.264/AVC bitstream. However, there are situations when I_PCM mode may be useful, for example:

1. When video is coded at a very high perceived quality / very low QP setting, for example, for applications such as content archiving or distribution where high quality is required, there may occasionally be macroblocks which, when coded using the usual processes of prediction, transform and entropy coding, generate **more** bits than the original 'raw' data.
2. If I_PCM mode is selected for every macroblock, the H.264/AVC bitstream becomes a 'container' for uncompressed video. This makes it possible for an application to make use of all the other features of H.264/AVC such as packet handling, frame numbering and transport support, whilst retaining the original video samples.

5.7.3 Macroblock prediction

The macroblock prediction syntax elements indicate how intra or inter prediction is carried out for the current macroblock. Sub_mb_pred is used for P or B macroblocks coded with 8×8 partition size and mb_pred is used for all other cases (Figure 5.14). For a detailed description of intra and inter prediction, see Chapter 6.

I macroblock: If the prediction type is 4×4 or 8×8, the prediction modes for each 4×4 or 8×8 luma block are signalled. Note that if the prediction type is 16×16, the mode has already been signalled as part of mb_type (section 5.7.1). The chroma prediction mode is signalled. Section 5.2.5 explains how these modes are coded.

B macroblock, Direct 16×16 mode: No further prediction information is sent; List0 and List1 reference frame indices and List 0 and List 1 motion vectors are derived from previously coded macroblocks (Chapter 6).

P or B macroblock, 8×8 partition size: For each of the four 8×8 partitions or sub-macroblocks, sub_mb_type is sent indicating for the sub-macroblock:

(a) Direct mode or
(b) Sub-macroblock partition size, 8×8, 8×4, 4×8 or 4×4, and prediction sources, List 0, List 1 and/or BiPredicted. This is followed by some or all of the following elements, depending on sub_mb_type:

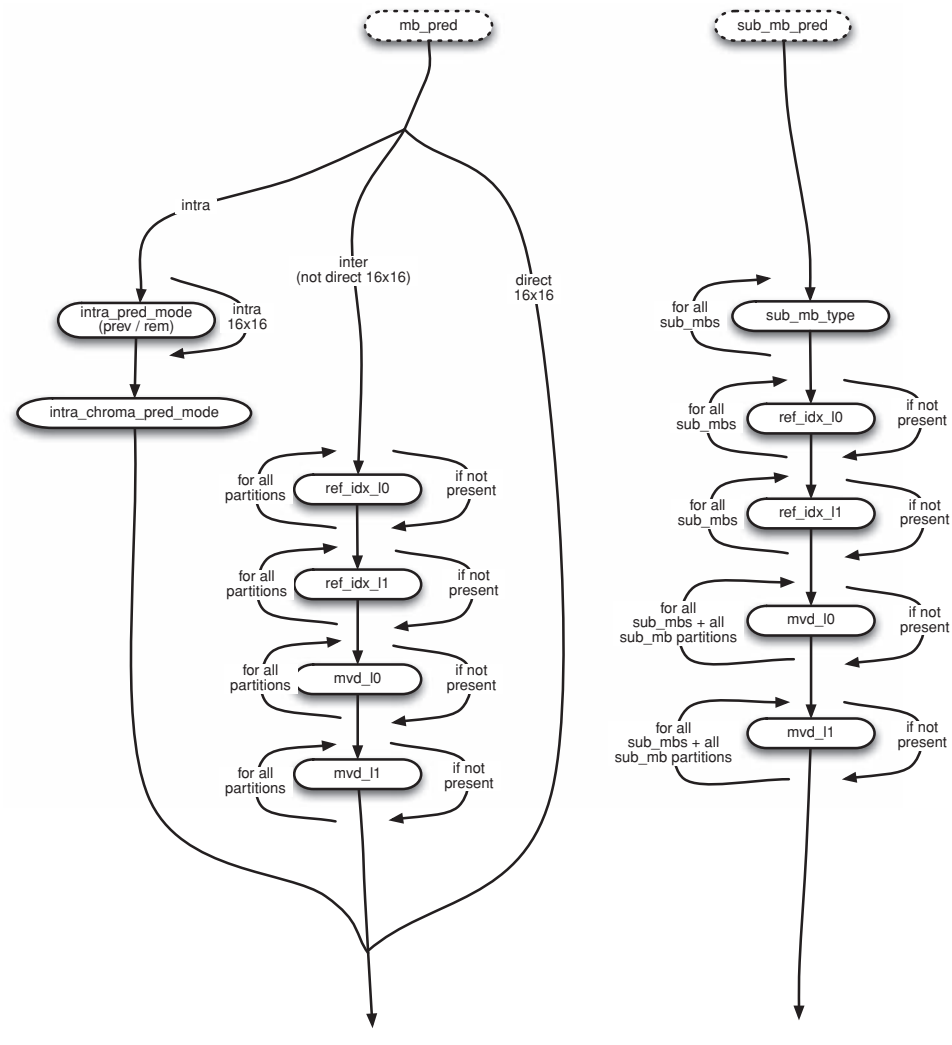


Figure 5.14 mb_pred and sub_mb_pred syntax overview

(i) Reference picture index/indices for List 0 and/or List 1, one index or one pair of indices for each sub-macroblock
(ii) Motion vector differences (mvd) for List 0 and/or List 1.

Note that all sub-macroblock partitions in a sub-macroblock share the same reference picture(s), whereas each sub-macroblock partition has its own motion vector (x,y) pair.

All other P or B macroblocks: Some or all of the following are sent for each macroblock partition, i.e. one 16×16 partition, two 8×16 partitions or two 16×8 partitions:

(a) Reference picture index/indices for List 0 and/or List 1
(b) Motion vector differences for List 0 and/or List 1.

Table 5.14 Type and prediction elements for B macroblocks, excluding Direct or 8 × 8 partitions

Partition	Type (part 0)	Type (part 1)	Number of reference indices	Number of mvd (x,y) pairs
16 × 16	L0 or L1	N/A	1	1
16 × 16	Bipred	N/A	2 : one for each List	2
16 × 8	L0 or L1	L0 or L1	2 : one for each partition	2
‘	L0 or L1	Bipred	3 : one for partition 0, two for partition 1	3
‘	Bipred	L0 or L1	3 : two for partition 0, one for partition 1	3
‘	Bipred	Bipred	4 : two for each partition	4
8 × 16	Same options as for 16 × 8			

The motion vector differences (mvd) are added to a predicted motion vector to create the x and y components of each motion vector (Chapter 6).

As an example, Table 5.14 lists the type and prediction elements, reference indices and motion vector difference (mvd) pairs, for B macroblocks excluding the special cases of Direct and 8 × 8 partition types. Each partition has one or two reference frames, one from List 0, one from List 1 or one from each List, i.e. biprediction, and a corresponding number of mvd(x,y) pairs signalled in the bitstream.

Note that each syntax element is only sent if it is needed. For example, if there is only one reference frame in a List, it is not necessary to send a reference index for that List and so the decoder does not expect to receive one.

5.7.4 Residual data

Residual data for a complete macroblock, if present, is sent according to the syntax summarised in Figure 5.15. Figure 5.16 shows the sequence in which residual blocks are coded. First, a 4 × 4 block of transformed luma DC coefficients is sent if Intra 16 × 16 mode was chosen for this macroblock. This optional first block is labelled ‘-1’ in the Figure 5.16. Then, the luma component is sent in 8 × 8 block order. If coded_block_pattern indicates the 8 × 8 block contains no coefficients (section 5.7.1), that block is bypassed. Each 8 × 8 luma block is processed using four 4 × 4 transforms (Figure 5.16) or one 8 × 8 transform (Figure 5.17, High Profiles only).

After all the coded luma transform coefficients are sent, two blocks of transformed chroma DC coefficients are transmitted, if coded_block_pattern indicates they are present, e.g. for blocks 16 and 17 in Figure 5.16, blocks 4 and 5 in Figure 5.17. Finally, the chroma AC blocks are transmitted if present. If the sampling format is 4:2:0 (Chapter 2), each chroma DC block contains 2 × 2 samples and each chroma AC block contains 8 × 8 samples. These sizes change if 4:2:2 or 4:4:4 sampling is employed (Chapter 7).

Each residual block is coded using CAVLC or CABAC. The CAVLC and CABAC block syntax is summarized in Figure 5.18 and Figure 5.19 and described in detail in Chapter 7.

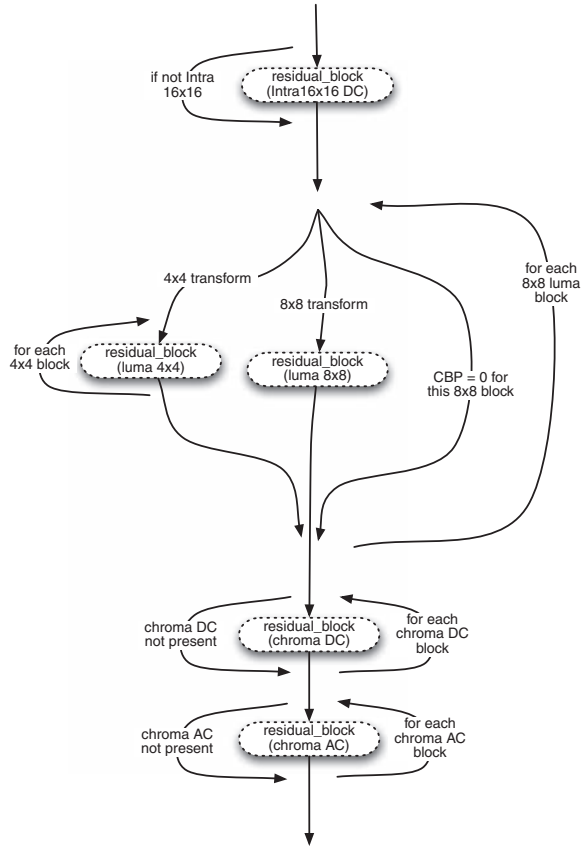


Figure 5.15 Residual data syntax overview

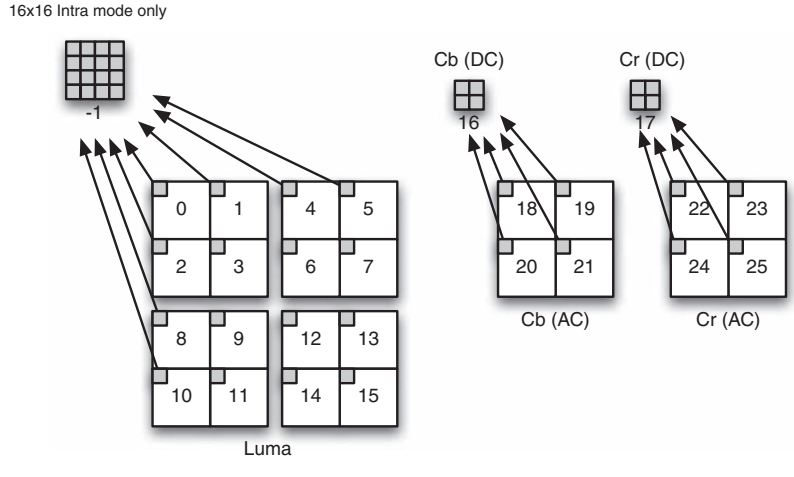


Figure 5.16 Block scanning order, 4×4 transform, 4:2:0 sampling

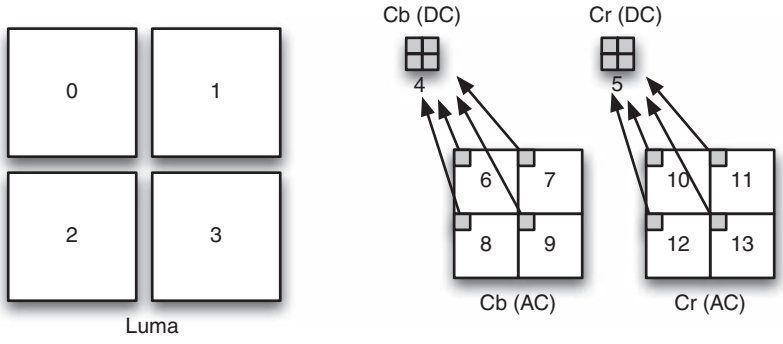


Figure 5.17 Block scanning order, 8 × 8 luma transform, 4:2:0 sampling

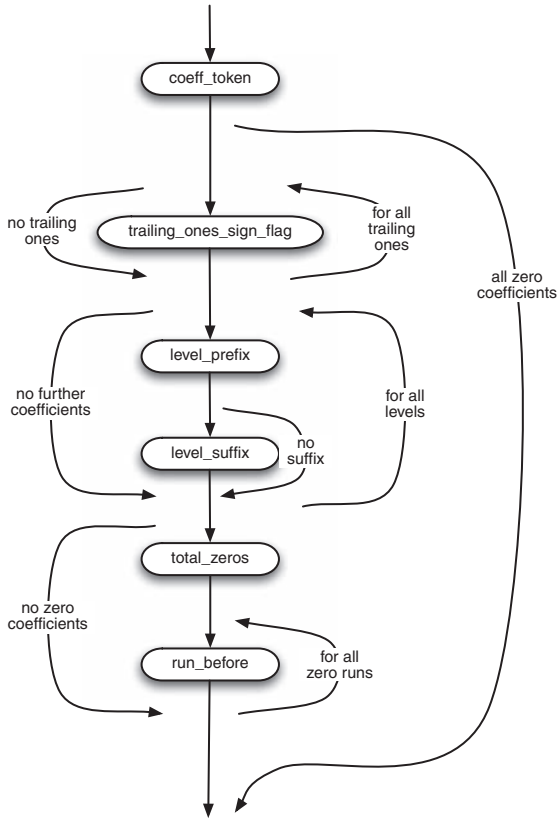


Figure 5.18 Residual CAVLC block syntax overview

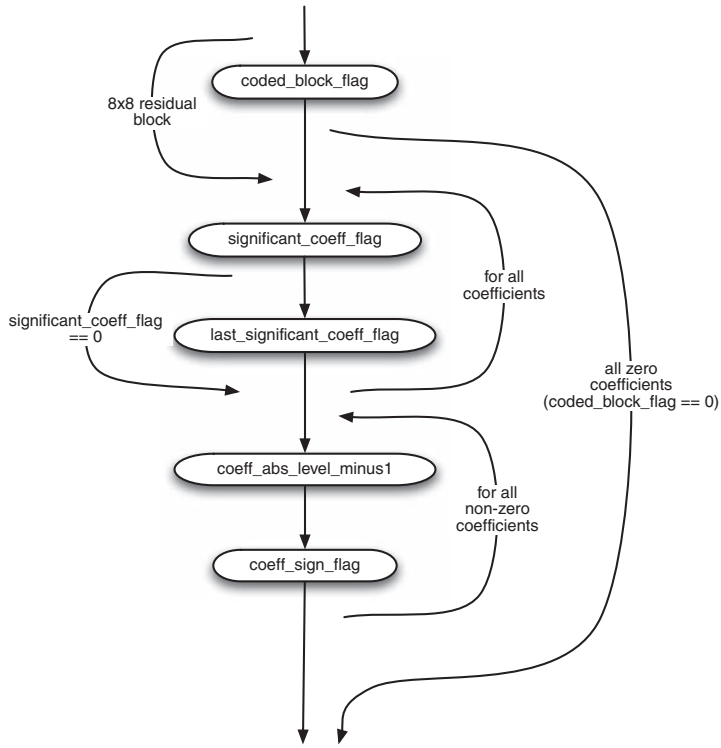


Figure 5.19 Residual CABAC block syntax overview

5.7.5 Macroblock syntax examples

The following examples are taken from trace files generated with the JM H.264 reference software codec [i].

1. Intra coded macroblock
   Table 5.15 shows the first part of a trace for an Intra coded macroblock. The first element, mb_type, indicates that this is an Intra macroblock using 4×4 prediction. The prediction modes of each 4×4 block are signalled, followed by the prediction mode of the chroma blocks. See Chapter 6 for a detailed description of how the intra mode is signalled using Most Probable and Rem parameters. Coded_block_pattern is $31_{10} = \mathbf{01}1111_2$, i.e. all luma blocks and the chroma DC blocks contain non-zero coefficients, but there are no coefficients present for chroma AC.

2. P macroblock, one reference frame
   In this example, illustrated in Table 5.16 and Figure 5.20, a P macroblock is coded using 16×16 mode, i.e. one motion vector per macroblock. There is only one reference frame and so the choice of reference is not signalled. MVDx $= 0$ and MVDy $= -1$, indicating

Table 5.15 I macroblock, example 1

Parameter	Binary code	Symbol	Discussion
mb_type (I_SLICE) (0, 1) = 9	1	0	Intra coding, 4×4 prediction of each luma block.
Intra mode = 3 0	0011	0,3	Rem = 3
Intra mode = −1 1	1	1	Most probable (MP)
Intra mode = 2 2	0010	0,2	Rem = 2
Intra mode = −1 3	1	1	MP
Intra mode = −1 4	1	1	MP
Intra mode = 2 5	0010	0,2	Rem = 2
Intra mode = 5 6	0101	0,5	Rem = 5
Intra mode = 5 7	0101	0,5	Rem = 5
Intra mode = 5 8	0101	0,5	Rem = 5
Intra mode = −1 9	1	1	MP
Intra mode = −1 10	1	1	MP
Intra mode = 1 11	0001	0,1	Rem = 1
Intra mode = −1 12	1	1	MP
Intra mode = 2 13	0010	0,2	Rem = 2
Intra mode = −1 14	1	1	MP
Intra mode = 3 15	0011	0,3	Rem = 3
Chroma intra pred mode	01	0	Mode 0
CBP (0, 1) = 31	10	31	Coefficients present for all luma blocks and for chroma DC blocks.
Delta QP (0, 1) = 0	1	0	No change in QP
Luma # c & tr.1s(0,0) vlc=2 #c=8 #t1=3	1101	8	Coefficient token (see Chapter 7)
Luma trailing ones sign (0,0)	0	0	Trailing ones
Luma lev (0,0) k=4 vlc=0 lev= 1	1	1	Level
Luma lev (0,0) k=3 vlc=1 lev= −1	11	−1	Level
Luma lev (0,0) k=2 vlc=1 lev= −2	11	−2	Level
Luma lev (0,0) k=1 vlc=1 lev= −4	11	−4	Level
Luma lev (0,0) k=0 vlc=2 lev= 3	100	3	Level
Luma totalrun (0,0) vlc=7 totzeros= 1	1	1	Total number of zeros
Luma run (0,0) k=7 vlc=0 run= 0	1	0	Zero run
Luma run (0,0) k=6 vlc=0 run= 1	0	1	Zero run
. . . (etc)			

Table 5.16 P macroblock, example 2

Parameter	Binary code	Symbol	Discussion
mb_skip_run	1	0	No preceding skip
mb_type (P_SLICE) (0, 2) = 1	1	0	16 × 16 mode
mvd_l0 (0) = 0 (org_mv 0 pred_mv 0)	1	0	X vector difference
mvd_l0 (1) = −1 (org_mv 0 pred_mv 1)	11	−1	Y vector difference
CBP (0, 2) = 1	11	1	000001_2: Only first block has coefficient data
Delta QP (0, 2) = 0	1	0	No change in QP
Luma # c & tr.1s(0,0) vlc=0 #c=3 #t1=2	101	3	Coefficient token
Luma trailing ones sign (0,0)	10	2	Trailing ones
Luma lev (0,0) k=0 vlc=0 lev= −2	1	−1	Level
Luma totalrun (0,0) vlc=2 totzeros= 2	110	2	Total zeros
Luma run (0,0) k=2 vlc=1 run= 1	1	1	Zero run
Luma run (0,0) k=1 vlc=0 run= 0	1	0	Zero run
Luma # c & tr.1s(1,0) vlc=1 #c=0 #t1=0	11	0	Zero 4 × 4 block
Luma # c & tr.1s(0,1) vlc=1 #c=0 #t1=0	11	0	‘
Luma # c & tr.1s(1,1) vlc=0 #c=0 #t1=0	1	0	‘

an offset of $-{}^1/_4$ sample in the y direction from the predicted motion vector (Chapter 6). The predicted vector is $(0,+{}^1/_4)$ so the actual motion vector is (0, 0). Only one 8 × 8 block, the top-left, contains non-zero coefficients. Within this 8 × 8 block, only the top-left 4 × 4 block contains non-zero coefficients. CBP indicates that coefficient information is only sent for the first 8 × 8 luma quadrant; non-zero coefficients are signalled for the first 4 × 4 block in this quadrant. The decoder fills the remaining 4 × 4 blocks with zeros.

3. P macroblock, one reference frame
   Another P macroblock example is shown in Table 5.17. The two preceding macroblocks are skipped. The macroblock uses 8 × 16 prediction mode with one reference frame. The MVD for the first or left-hand partition is $(0, {}^1/_4)$ and MVD for the second or right-hand

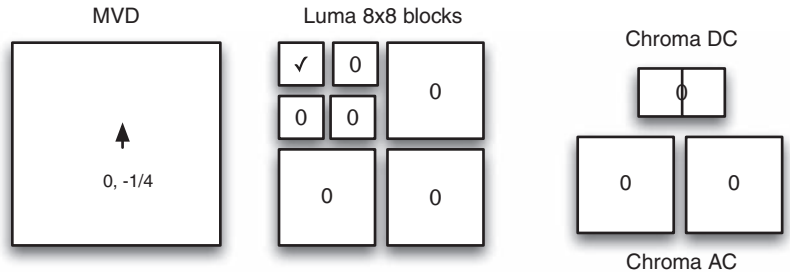


Figure 5.20 P macroblock, example 2

Table 5.17 P macroblock, example 3

Parameter	Binary code	Symbol	Discussion
mb_skip_run	11	2	Two previous MBs are skipped
mb_type (P_SLICE) (0, 3) = 3	11	2	8×16 mode
mvd_l0 (0) = 0	1	0	Partition 0 MVDx
mvd_l0 (1) = 1	10	1	Partition 0 MVDy
mvd_l0 (0) = −1	11	−1	Partition 1 MVDx
mvd_l0 (1) = 0	1	0	Partition 1 MVDy
CBP (0, 3) = 1	11	1	000001_2: Only first 8×8 block non zero
Delta QP (0, 3) = 0	1	0	No change in QP
Luma # c & tr.1s(0,0) vlc=0 #c=0 #t1=0	1	0	Zero 4×4 block
Luma # c & tr.1s(1,0) vlc=0 #c=1 #t1=1	1	1	Coefficient token
Luma trailing ones sign (1,0)	1	1	Trailing ones
Luma totalrun (1,0) vlc=0 totzeros= 1	11	1	Total zeros
Luma # c & tr.1s(0,1) vlc=0 #c=2 #t1=2	1	2	Coefficient token
Luma trailing ones sign (0,1)	10	2	Trailing ones
Luma totalrun (0,1) vlc=1 totzeros= 3	100	3	Total zeros
Luma run (0,1) k=1 vlc=2 run= 1	10	1	Zero run
Luma # c & tr.1s(1,1) vlc=1 #c=0 #t1=0	11	0	Zero 4×4 block

partition is ($-1/4$, 0). Only the first, top-left 8×8 luma quadrant contains non-zero coefficients and the second and third 4×4 blocks within this quadrant contain non-zero coefficients.

4. P macroblock, multiple reference frames
   In this example (Table 5.18, Figure 5.21) the macroblock is coded using 8×8 partitions. A further syntax element sub_mb_type is sent for each of the four 8×8 luma partitions to indicate the sub-macroblock partition types. Next, the reference frame for each macroblock partition is signalled. Note that all sub-macroblock partitions in a partition share the same reference frame. Partition 0 uses reference index 2, three frames in the past, partition 1 uses reference index 1, two frames in the past, and partitions 2 and 3 use reference index 0, one frame in the past, the default case. MVDx and MVDy are sent for each partition or sub-macroblock partition; the only non-zero values are (−3, 0) and (−3, 1). Note that these are signalled as (−12,0) and (−12, 4) respectively, in units of $1/4$ sample. CBP indicates that the only non-zero coefficients occur in the Chroma DC blocks.

5. B macroblock
   This B macroblock is coded using two 8×16 partitions (Table 5.19, Figure 5.22). Partition 0 is predicted from the first picture in List 1, the next frame in display order, and partition 1 is bi-predicted from the first pictures in List 0 and List 1, the previous

Table 5.18 P macroblock, example 4

Parameter	Binary code	Symbol	Discussion
mb_skip_run	1	0	No preceding skipped MBs
mb_type (P_SLICE) (5, 2) = 8	100	3	8×8 partitions with multiple reference frames
sub_mb_type(0)	1	0	Partition 0 mode = 8×8
sub_mb_type(1)	1	0	Partition 1 mode = 8×8
sub_mb_type(2)	10	1	Partition 2 mode = 8×4
sub_mb_type(3)	10	1	Partition 3 mode = 8×4
ref_idx_l0 = 2	11	2	Part 0 L0 ref = 2 (3 prev)
ref_idx_l0 = 1	10	1	Part 1 L0 ref = 1 (2 prev)
ref_idx_l0 = 0	1	0	Part 2 L0 ref = 0 (prev)
ref_idx_l0 = 0	1	0	Part 3 L0 ref = 0 (prev)
mvd_l0 (0) = 0	1	0	Part 0 MVDx
mvd_l0 (1) = 0	1	0	Part 0 MVDy
mvd_l0 (0) = 0	1	0	Part 1 MVDx
mvd_l0 (1) = 0	1	0	Part 1 MVDy
mvd_l0 (0) = −12	11001	−12	Part 2 sub 0 MVDx
mvd_l0 (1) = 0	1	0	Part 2 sub 0 MVDy
mvd_l0 (0) = 0	1	0	Part 2 sub 1 MVDx
mvd_l0 (1) = 0	1	0	Part 2 sub 1 MVDy
mvd_l0 (0) = 0	1	0	Part 3 sub 0 MVDx
mvd_l0 (1) = 0	1	0	Part 3 sub 0 MVDy
mvd_l0 (0) = −12	11001	−12	Part 3 sub 1 MVDx
mvd_l0 (1) = 4	1000	4	Part 3 sub 1 MVDy
CBP (5, 2) = 16	10	16	010000_2 : Only Chroma DC present
Delta QP (5, 2) = 0	1	0	No change in QP
ChrDC # c & tr.1s(0,0) vlc=0 #c=1 #t1=1	1	1	Cb DC coefficients
ChrDC trailing ones sign (0,0)	0	0	Just a single trailing one
ChrDC totalrun (0,0) vlc=0 totzeros= 2	1	2	Zero run
ChrDC # c & tr.1s(1,1) vlc=0 #c=0 #t1=0	1	0	No Cr DC coefficients

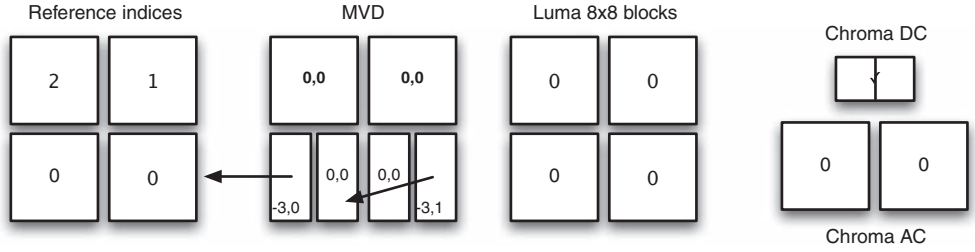


Figure 5.21 P macroblock example 4

Table 5.19 B macroblock, example 5

Parameter	Binary code	Symbol	Discussion
mb_skip_run	1	0	No preceding skipped MBs
mb_type (B_SLICE) (5, 4) = 3	10000	15	B_L1_Bi_8 × 16 : partition 0 from L1, partition 1 is bipred
ref_idx_l0 = 0	1	0	Part 1 L0 ref, order: all L0s then all L1s in partition order
ref_idx_l1 = 0	1	0	Part 0 L1 ref
ref_idx_l1 = 0	1	0	Part 1 L1 ref
mvd_l0 (0) = 11	10110	11	Part 1 L0 MVDx
mvd_l0 (1) = 2	100	2	Part 1 L0 MVDy
mvd_l1 (0) = −3	111	−3	Part 0 L1 MVDx
mvd_l1 (1) = 0	1	0	Part 0 L1 MVDy
mvd_l1 (0) = 1	10	1	Part 1 L1 MVDx
mvd_l1 (1) = −2	101	−2	Part 1 L1 MVDy
CBP (5, 4) = 0	1	0	000000_2: No coefficient data

+ next frames in display order. Partition 0 has MVD = ($-^3/_4$, 0) which, when added to the predicted vector, indicates an offset of ($7^3/_4$, $-1^1/_4$) from the reference frame, the next frame in display order. Partition 1 has two MVD pairs, ($2^3/_4$,$^1/_2$) with respect to the previous frame and ($^1/_4$,$-^1/_2$) with respect to the next frame in display order. The final motion vectors and reference regions are shown in Figure 5.22. There are no coded coefficients; this is quite common for B macroblocks, especially at lower bitrates, because inter prediction in a B macroblock tends to be very efficient, leading to minimal or zero energy in the quantized residual.

6. B macroblock
   Another B macroblock example is shown in Table 5.20. Similarly to the previous example, there are two 8 × 16 partitions. This time, the first is predicted from L0, the previous frame, and the second is bipredicted using L0, the previous frame, and L1, the next frame. Compare the order of the reference indices and motion vector differences with the previous example.

7. B macroblock
   This macroblock (Table 5.21) has a single 16 × 16 partition, bipredicted from the previous (L0) and next (L1) reference frames. One MVD pair (MVDx, MVDy) is sent for each prediction reference.

8. B macroblock
   This example is a macroblock coded using four 8 × 8 macroblock partitions (Table 5.22). Partitions 0 and 2 are predicted using Direct Prediction (Chapter 6) and so no MVD or reference index is sent for these partitions. Partition 1 is a single 8 × 8 sub-macroblock partition, predicted from L1, the next reference frame. Partition 3 contains four 4 × 4 sub-macroblock partitions, all predicted from L1. Each of these 4 × 4 sub MB partitions has a separate MVDx, MVDy pair.

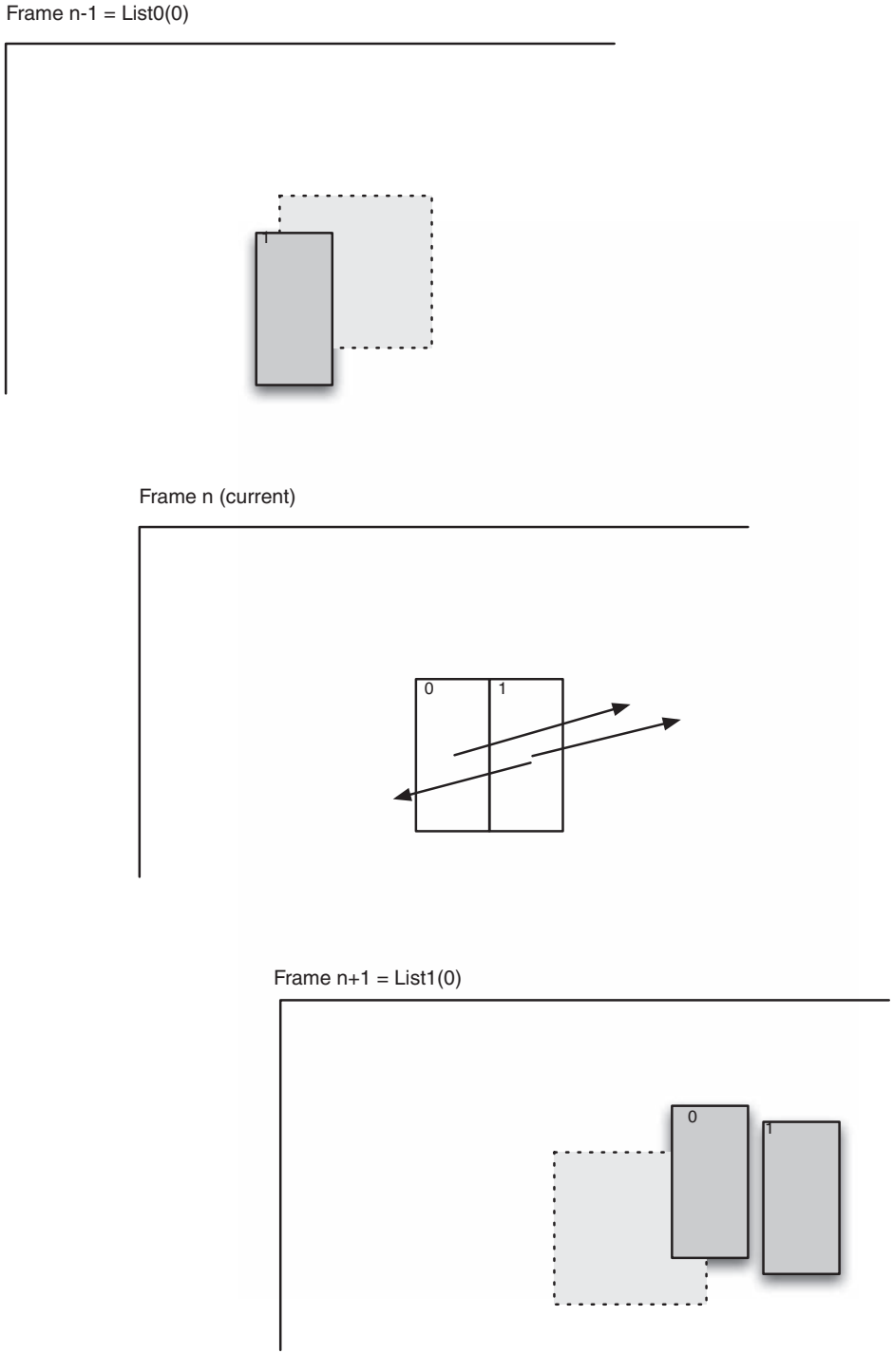


Figure 5.22 B macroblock, example 5

Table 5.20 B macroblock, example 6

Parameter	Binary code	Symbol	Discussion
mb_skip_run	1	0	No previous skip
mb_type (B_SLICE) (7, 2) = 3	1110	13	B_L0_Bi_8 × 16 : part 0 from L0, part 1 is bipred
ref_idx_l0 = 0	1	0	Part 0, L0 reference
ref_idx_l0 = 0	1	0	Part 1, L0 reference
ref_idx_l1 = 0	1	0	Part 1, L1 reference
mvd_l0 (0) = −22	101101	−22	Part 0, L0 MVDx
mvd_l0 (1) = 4	1000	4	Part 0, L0 MVDy
mvd_l0 (0) = 10	10100	10	Part 1, L0 MVDx
mvd_l0 (1) = 0	1	0	Part 1, L0 MVDy
mvd_l1 (0) = 11	10110	11	Part 1, L1 MVDx
mvd_l1 (1) = −1	11	−1	Part 1, L1 MVDy
CBP (7, 2) = 0	1	0	000000_2 : No coefficient data

5.8 Summary

At the heart of the H.264/AVC standard is the syntax, a specific format for representing a coded video sequence, together with ancillary information. A decoder parses the syntax, extracts the parameters and data elements and can then proceed to decode and display video. The syntax is organized hierarchically, from a complete video sequence at the highest level, down to coded macroblocks and blocks. A key feature of H.264/AVC is its highly flexible handling of pictures and predictions. Picture management procedures make it possible to construct and use a very large variety of prediction structures for intra and inter prediction. This flexibility makes it possible to accurately predict or estimate the current section of an image, which in turn leads to high compression efficiency. In the following chapters we will examine the prediction of macroblocks, transform and quantization of residuals and coding of syntax elements.

Table 5.21 B macroblock, example 7

Parameter	Binary code	Symbol	Discussion
mb_skip_run	110	5	5 previous MBs are skipped.
mb_type (B_SLICE) (7, 2) = 1	100	3	B_Bi_16 × 16 : one partition, bipredicted
ref_idx_l0 = 0	1	0	L0 reference
ref_idx_l1 = 0	1	0	L1 reference
mvd_l0 (0) = −1	11	−1	L0 MVDx
mvd_l0 (1) = 0	1	0	L0 MVDy
mvd_l1 (0) = −1	11	−1	L1 MVDx
mvd_l1 (1) = −1	11	−1	L1 MVDy
CBP (7, 2) = 0	1	0	No coefficient data

Table 5.22 B macroblock, example 8

Parameter	Binary code	Symbol	Discussion
mb_skip_run	10	1	1 previous MB is skipped
mb_type (B_SLICE) (6, 6) = 8	10111	22	B_8 × 8
sub_mb_pred(0)	1	0	Direct, no MVD or ref_ix
sub_mb_pred(1)	11	2	B_L1_8 × 8, 8 × 8 predicted from L1
sub_mb_pred(2)	1	0	Direct, no MVD or ref_ix
sub_mb_pred(3)	1100	11	B_L1_4 × 4, 4 × 4 predicted from L1
ref_idx_l1 = 0	1	0	L1 reference for 8 × 8 block 1
ref_idx_l1 = 0	1	0	L1 reference for 8 × 8 block 3
mvd_l1 (0) = −6	1101	−6	Block 1 MVDx
mvd_l1 (1) = 1	10	1	Block 1 MVDy
mvd_l1 (0) = 0	1	0	Block 3, subblock 0, MVDx
mvd_l1 (1) = 0	1	0	Block 3, subblock 0, MVDy
mvd_l1 (0) = 0	1	0	Block 3, subblock 1, MVDx
mvd_l1 (1) = 0	1	0	Block 3, subblock 1, MVDy
mvd_l1 (0) = 0	1	0	Block 3, subblock 2, MVDx
mvd_l1 (1) = −3	111	−3	Block 3, subblock 2, MVDy
mvd_l1 (0) = 0	1	0	Block 3, subblock 3, MVDx
mvd_l1 (1) = 0	1	0	Block 3, subblock 3, MVDy
CBP (6, 6) = 22	101101	22	010110_2 : Luma coefficients in 8 × 8 blocks 1 and 2. Chroma DC coefficients present, no chroma AC coefficients.
Delta QP (6, 6) = 0	1	0	No change in QP.
Luma # c & tr.1s(2,0) vlc=0 #c=0 #t1=0	1	0	First 4 × 4 block of luma 8 × 8 block 1. . .
. . ...further data for luma 8 × 8 blocks 1 and 2 and chroma DC coefficients.			

5.9 References

i. JM reference software version 16.0, http://iphome.hhi.de/suehring/tml/, July 2009.
ii. T. Wiegand, G. J. Sullivan, G. Bjontegaard and A. Luthra, 'Overview of the H.264/AVC video coding standard', *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7. (4 August 2003), pp. 560–576.
iii. L. Wang, R. Gandhi, K. Panusopone, Y. Yu and A. Luthra, 'MB-Level Adaptive Frame/Field Coding for JVT', Joint Video Team Document JVT-B106, January 2002.

6

H.264 Prediction

6.1 Introduction

Perhaps the most important reason for the widespread adoption of H.264/AVC is its compression performance. An H.264 codec, if properly designed and used, can out-perform many other video codecs in terms of compression ratio for a given image quality. Much of the performance gain compared with previous standards is due to H.264/AVC's efficient prediction methods. For every macroblock, a prediction is created, an attempt to duplicate the information contained in the macroblock using previously coded data, and subtracted from the macroblock to form a residual. The efficiency or accuracy of this prediction process has a significant impact on compression performance. An accurate prediction means that the residual contains very little data and this in turn leads to good compression performance.

H.264/AVC supports a wide range of prediction options – intra prediction using data within the current frame, inter prediction using motion compensated prediction from previously coded frames, multiple prediction block sizes, multiple reference frames and special modes such as Direct and Weighted prediction. Together with sub-pixel interpolation and a built-in filter to reduce compression artefacts, these features give an H.264 encoder a great deal of flexibility in the prediction process. By selecting the best prediction options for an individual macroblock, an encoder can minimize the residual size to produce a highly compressed bitstream.

We start with an overview of macroblock prediction, followed by a detailed look at intra and inter prediction in H.264/AVC.

6.2 Macroblock prediction

Figure 6.1 shows the prediction sources for three macroblocks, an I Macroblock, a P Macroblock and a B Macroblock. An I Macroblock (I MB) is predicted using intra prediction from neighbouring samples in the current frame. A P Macroblock (P MB) is predicted from samples in a previously-coded frame which may be before or after the current picture in display order, i.e. a 'past' or a 'future' frame. Different rectangular sections (**partitions**) in a P MB may

The H.264 Advanced Video Compression Standard, Second Edition Iain E. Richardson
© 2010 John Wiley & Sons, Ltd

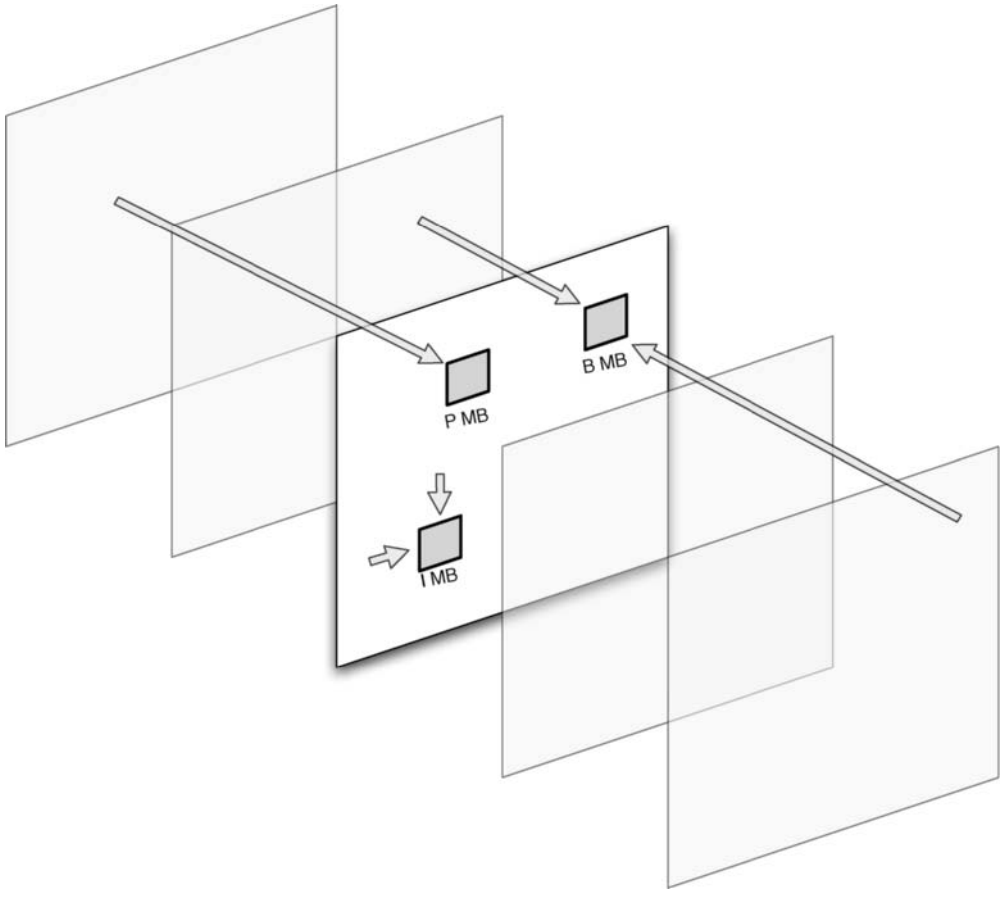


Figure 6.1 Example of macroblock types and prediction sources

be predicted from different reference frames. Each partition in a B Macroblock (B MB) is predicted from samples in one or two previously-coded frames, for example, one 'past' and one 'future' as shown in the figure.

6.3 Intra prediction

An intra (I) macroblock is coded without referring to any data outside the current slice. I macroblocks may occur in any slice type. Every macroblock in an I slice is an I macroblock. I macroblocks are coded using intra prediction, i.e. prediction from previously-coded data in the same slice. For a typical block of luma or chroma samples, there is a relatively high correlation between samples in the block and samples that are immediately adjacent to the block. Intra prediction therefore uses samples from adjacent, previously coded blocks to predict the values in the current block.

Example

Figure 6.2 shows 4 × 4 blocks in the luma component of a macroblock. The current block is number 10 in the figure. Blocks 0-9 have already been coded and transmitted and are therefore available to the decoder by the time block 10 is decoded. This means that any of the samples in blocks 0 to 9 are potential candidates for intra prediction. However, an H.264 encoder may only choose certain samples in blocks **1, 3, 5** and **8** to generate a 4 × 4 intra prediction.

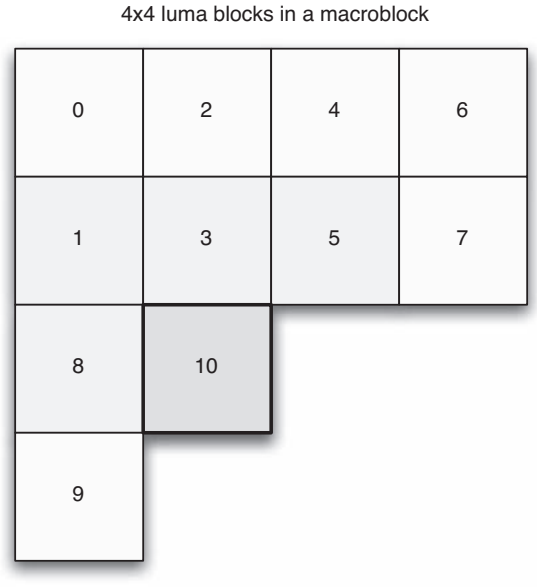


Figure 6.2 Intra prediction: adjacent blocks example

In an intra macroblock, there are three choices of intra prediction block size for the luma component, namely 16 × 16, 8 × 8 or 4 × 4. A single prediction block is generated for each chroma component. Each prediction block is generated using one of a number of possible prediction modes (Table 6.1).

Table 6.1 Intra prediction types

Intra prediction block size	Notes
16 × 16 (luma)	A single 16 × 16 prediction block P is generated. Four possible prediction modes.
8 × 8 (luma)	An 8 × 8 prediction block P is generated for each 8 × 8 luma block. Nine possible prediction modes. 'High' Profiles only.
4 × 4 (luma)	A 4 × 4 prediction block P is generated for each 4 × 4 luma block. Nine possible prediction modes.
Chroma	One prediction block P is generated for each chroma component. Four possible prediction modes. The same prediction mode is used for both chroma components.

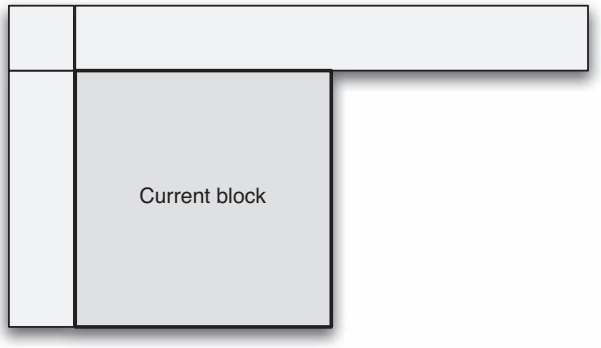


Figure 6.3 Intra prediction source samples, 4 × 4 or 8 × 8 luma blocks

When 4 × 4 or 8 × 8 block size is chosen for the luma component, the intra prediction is created from samples directly above the current block; directly to the left; above and to the left; above and to the right; or a combination of these depending on the current prediction mode and on whether the required samples have already been coded (Figure 6.3). For a 16 × 16 luma block or a chroma block, the prediction is created from samples directly to the left or above the current block or from a combination of these (Figure 6.4).

Only those samples that are actually available may be used to form a prediction. For example, a block on the left margin of the picture or slice has no neighbouring samples on the left side and so certain intra prediction modes are not available to the encoder. The encoder chooses an intra mode for the current block from the available prediction modes. The choice of intra mode is communicated to the decoder as part of the coded macroblock.

The choice of intra prediction block size for the luma component, 16 × 16, 4 × 4 or 8 × 8 when available, tends to be a trade-off between (i) prediction efficiency and (ii) cost of signalling the prediction mode.

(a) Smaller blocks: A smaller prediction block size (4 × 4) tends to give a more accurate prediction, i.e. the intra prediction for each block is a good match to the actual data in

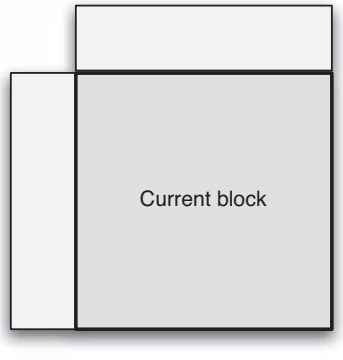


Figure 6.4 Intra prediction source samples, chroma or 16 × 16 luma blocks

the block. This in turn means a smaller coded residual, so that fewer bits are required to code the quantized transform coefficients for the residual blocks. However, the choice of prediction for every 4 × 4 block must be signalled to the decoder, which means that more bits tend to be required to code the prediction choices.

(b) Larger blocks: A larger prediction block size (16 × 16) tends to give a less accurate prediction, hence more residual data, but fewer bits are required to code the prediction choice itself.

An encoder will typically choose the appropriate intra prediction mode to minimize the total number of bits in the prediction and the residual.

Example 1:

Figure 6.5 shows a typical I picture from a CIF sequence coded using the Baseline profile, with the prediction block sizes superimposed. In homogeneous regions of the frame, where the texture is largely uniform, 16 × 16 prediction mode tends to be more efficient since the prediction is reasonably accurate and the prediction mode overhead is low. In more complex regions of the frame, 4 × 4 mode is often selected because the increased rate required to signal the prediction mode is offset by the reduced residual size.

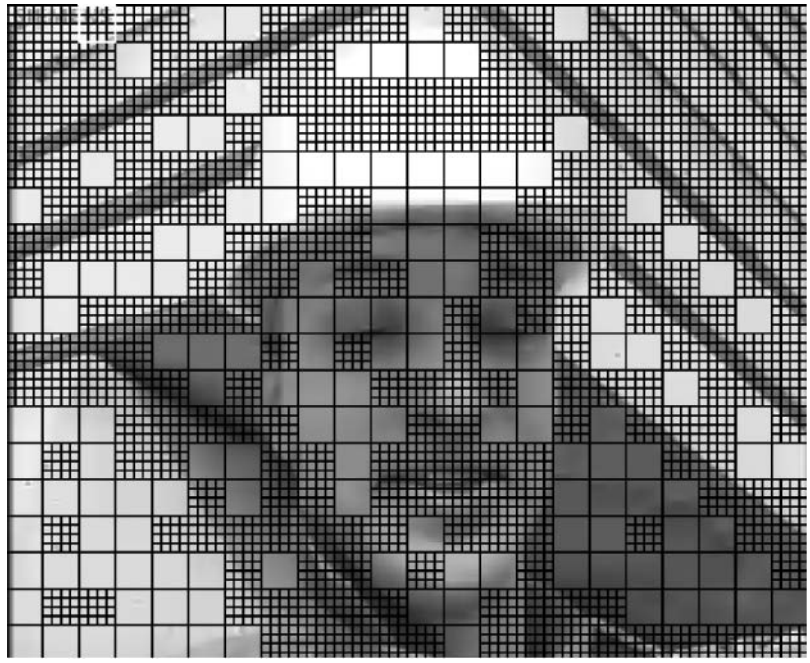


Figure 6.5 Example of intra block size choices, CIF, Baseline Profile. Reproduced by permission of Elecard.

Example 2:

A QCIF video frame (Figure 6.6) is encoded in intra mode and each block or macroblock is predicted from neighbouring, previously-encoded samples. Figure 6.7 shows the predicted

Figure 6.6 QCIF frame with highlighted macroblock

Figure 6.7 Predicted luma frame formed using H.264 intra prediction

luma frame *P* formed by choosing the best 4 × 4 or 16 × 16 prediction mode for each region. The predicted macroblocks (Figure 6.7) are not an accurate match for the original macroblocks (Figure 6.6). However, the predicted frame provides a rough approximation of the original. When the prediction is subtracted from the original, the residual (Figure 6.8) contains less information than the original frame and is therefore 'easier' to compress.

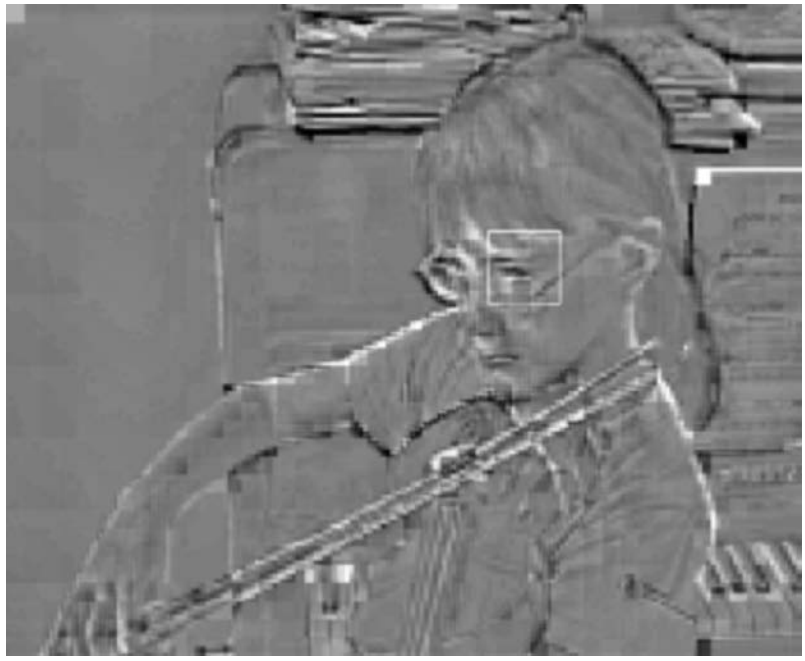


Figure 6.8 Residual after subtracting intra prediction

6.3.1 4 × 4 luma prediction modes

Figure 6.9 shows a 4 × 4 luma block, part of the highlighted macroblock in Figure 6.6, that is required to be predicted. The samples above and to the left, labelled A-M in Figure 6.10, have previously been encoded and reconstructed and are therefore available in the encoder and decoder to form a prediction reference. The samples a, b, c, . . ., p of the prediction block P (Figure 6.10) are calculated based on the samples A-M as follows. The method of forming a DC prediction, mode 2, is modified depending on which samples A-M are available and each of the other modes may only be used if all of the required prediction samples are available. However, note that if samples E, F, G and H are not available, the value of sample D is copied to these positions and they are marked as 'available'.

The arrows in Figure 6.11 indicate the direction of prediction in each mode. For modes 3-8, the predicted samples are formed from a weighted average of the prediction samples A-M. For example, if mode 4 is selected, the top-right sample of P, labelled 'd' in Figure 6.10, is predicted by: d = round(B/4 + C/2 + D/4).

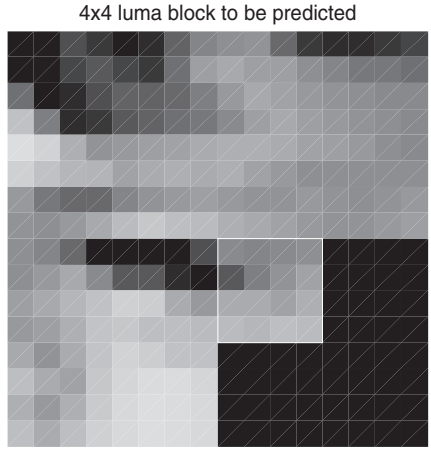


Figure 6.9 4 × 4 luma block to be predicted

M	A	B	C	D	E	F	G	H
I	a	b	c	d				
J	e	f	g	h				
K	i	j	k	l				
L	m	n	o	p				

Mode 0 (Vertical)	The upper samples A,B,C,D are extrapolated vertically.
Mode 1 (Horizontal)	The left samples I,J,K,L are extrapolated horizontally.
Mode 2 (DC)	All samples in P are predicted by the mean of samples A..D and I..L.
Mode 3 (Diagonal Down-Left)	The samples are interpolated at a 45° angle between lower-left and upper-right.
Mode 4 (Diagonal Down-Right)	The samples are extrapolated at a 45° angle down and to the right.
Mode 5 (Vertical-Left)	Extrapolation at an angle of approximately 26.6° to the left of vertical, i.e. width/height = $^1/_2$.
Mode 6 (Horizontal-Down)	Extrapolation at an angle of approximately 26.6° below horizontal.
Mode 7 (Vertical-Right)	Extrapolation or interpolation at an angle of approximately 26.6° to the right of vertical.
Mode 8 (Horizontal-Up)	Interpolation at an angle of approximately 26.6° above horizontal.

Figure 6.10 Labelling of prediction samples, 4 × 4 prediction

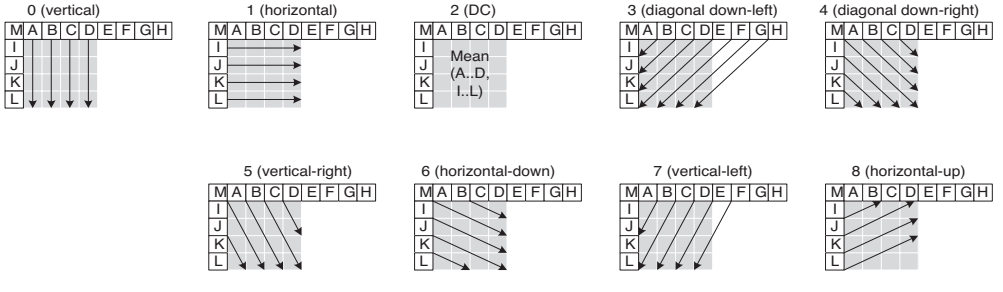


Figure 6.11 4 × 4 intra prediction modes

Example

The 9 prediction modes 0-8 are calculated for the 4 × 4 block shown in Figure 6.9 and the resulting prediction blocks P are shown in Figure 6.12. The Sum of Absolute Errors (SAE) for each prediction indicates the magnitude of the prediction error. In this case, the best match to

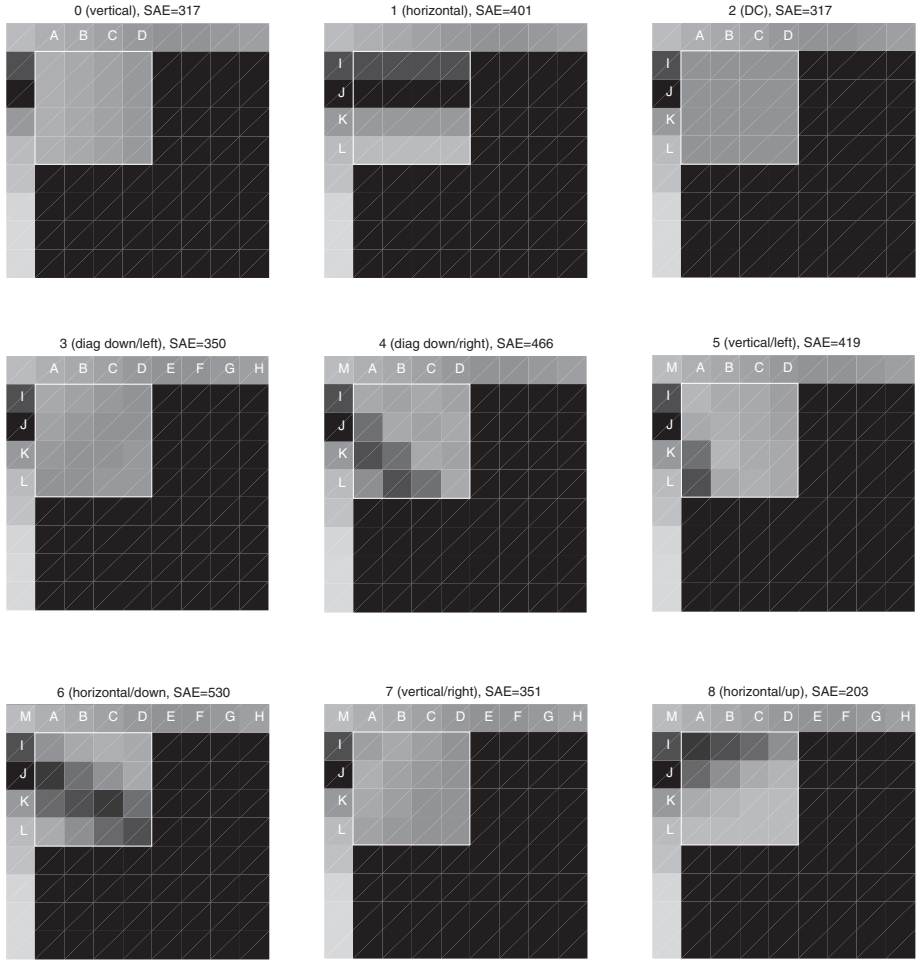


Figure 6.12 Prediction blocks, 4 × 4 modes 0−8

the actual current block is given by **mode 8**, horizontal-up, because this mode gives the smallest SAE. A visual comparison shows that the mode 8 P block appears quite similar to the original 4 × 4 block.

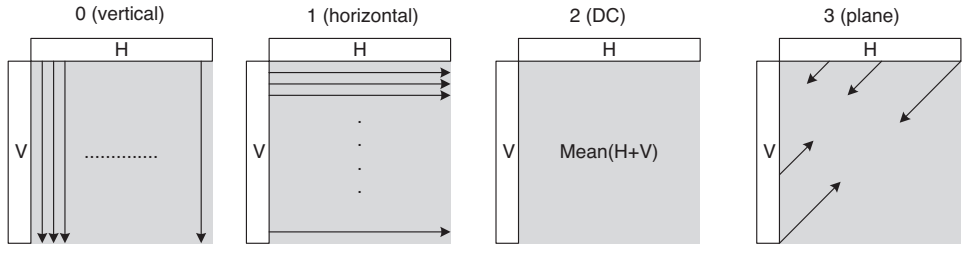


Figure 6.13 Intra 16 × 16 prediction modes

6.3.2 16 × 16 luma prediction modes

As an alternative to the 4 × 4 luma modes described above, the entire 16 × 16 luma component of a macroblock may be predicted in one operation. Four modes are available, shown in diagram form in Figure 6.13:

Mode 0 (vertical)	Extrapolation from upper samples (H)
Mode 1 (horizontal)	Extrapolation from left samples (V)
Mode 2 (DC)	Mean of upper and left-hand samples (H+V).
Mode 4 (Plane)	A linear 'plane' function is fitted to the upper and left-hand samples H and V. This works well in areas of smoothly-varying luminance.

Example

Figure 6.14 shows a luma macroblock with previously-encoded samples at the upper and left-hand edges. The results of the four prediction modes, shown in Figure 6.15, indicate that the best match is given by mode 3 which in this case produces a plane with a luminance gradient

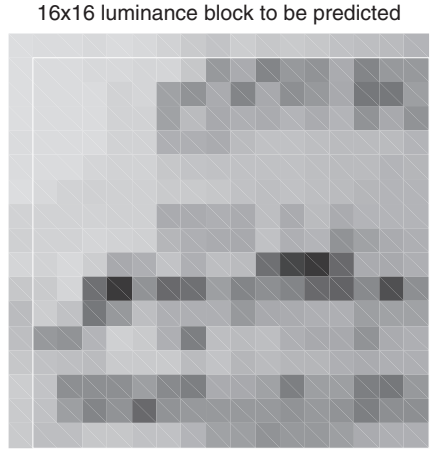


Figure 6.14 16 × 16 macroblock

from light at the upper-left to dark at the lower-right. Intra 16 × 16 mode tends to work best in homogeneous areas of an image.

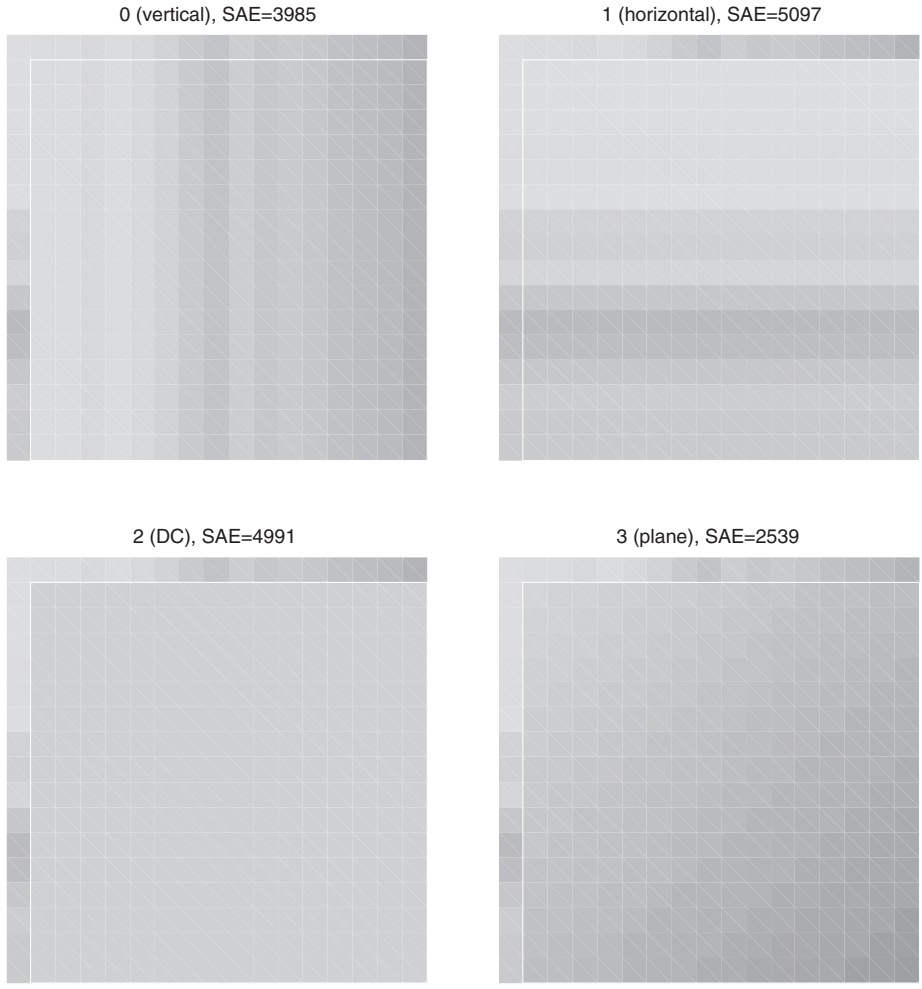


Figure 6.15 Prediction blocks, intra 16 × 16 modes 0−3

6.3.3 Chroma prediction modes

Each chroma component of a macroblock is predicted from previously encoded chroma samples above and/or to the left, with both chroma components always using the same prediction mode. The four prediction modes are very similar to the 16 × 16 luma prediction modes described in section 6.3.2 and illustrated in Figure 6.13, except that the numbering of

the modes is different. The modes are DC (mode 0), horizontal (mode 1), vertical (mode 2) and plane (mode 3).

6.3.4 8 × 8 luma prediction, High profiles

Intra prediction of the luma component with an 8 × 8 block size is only available in the High profiles (Chapter 8). Each 8 × 8 luma block in a macroblock is predicted using one of nine prediction modes which are very similar to the nine modes described in section 6.3.1 and illustrated in Figure 6.11.

6.3.5 Signalling intra prediction modes

6.3.5.1 4 × 4 or 8 × 8 luma prediction

The choice of intra prediction mode for each 4 × 4 or 8 × 8 luma block must be signalled to the decoder and this could potentially require a large number of bits. However, intra modes for neighbouring 4 × 4 or 8 × 8 blocks are highly correlated. For example, let A, B and E be the left, upper and current 4 × 4 blocks respectively (Figure 6.16). If previously-encoded 4 × 4 blocks A and B are predicted using mode 2, it is probable that the best mode for block E, the current block, is also mode 2. To take advantage of this correlation, predictive coding is used to signal 4 × 4 or 8 × 8 intra modes. This will be described for 4 × 4 intra modes; the method is similar for 8 × 8 intra modes.

For each current block E, the encoder and decoder calculate the most probable prediction mode, defined as the smaller of the prediction modes of A and B. If either of these neighbouring blocks is not available, i.e. outside the current slice or not coded in Intra4 × 4 mode, the corresponding value A or B is set to 2, indicating DC prediction mode.

The encoder sends a flag for each 4 × 4 block, *prev_intra4×4_pred_mode*. If the flag is '1', the most probable prediction mode is used. If the flag is '0', another parameter *rem_intra4×4_pred_mode* is sent to indicate a change of mode. If *rem_intra4×4_pred_mode* is smaller than the current most probable mode then the prediction mode is set to *rem_intra4×4_pred_mode*, otherwise the prediction mode is set to (*rem_intra4×4_pred_mode* +1). In this way, only eight values of *rem_intra4×4_pred_mode* are required, 0 to 7, to signal nine possible intra modes, 0 to 8.

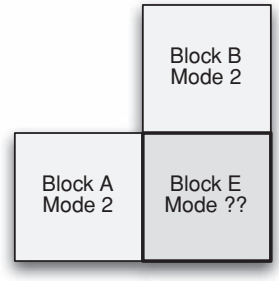


Figure 6.16 Intra mode prediction example

Example

Blocks A and B were predicted using modes 3, diagonal down-left, and 1, horizontal, respectively. The most probable mode for block E is therefore 1, horizontal. *prev_intra4×4_pred_mode* is set to '0' and so *rem_intra4×4_pred_mode* is sent. Depending on the value of *rem_intra4×4_pred_mode*, one of the eight remaining prediction modes listed in Table 6.2 may be chosen.

Table 6.2 Choice of prediction mode, most probable mode = 1

rem_intra4×4_pred_mode	Prediction mode for block E
0	0
1	2
2	3
3	4
4	5
5	6
6	7
7	8

16 × 16 luma prediction or chroma prediction
The prediction mode is signalled as part of the macroblock syntax and predictive mode coding is not used.

6.4 Inter prediction

Inter prediction is the process of predicting a block of luma and chroma samples from a picture that has previously been coded and transmitted, a reference picture. This involves selecting a prediction region, generating a prediction block and subtracting this from the original block of samples to form a residual that is then coded and transmitted. The block of samples to be predicted, a macroblock partition or sub-macroblock partition, can range in size from a complete macroblock, i.e. 16 × 16 luma samples and corresponding chroma samples, down to a 4 × 4 block of luma samples and corresponding chroma samples.

The reference picture is chosen from a list of previously coded pictures, stored in a Decoded Picture Buffer, which may include pictures before and after the current picture in display order (Chapter 5). The offset between the position of the current partition and the prediction region in the reference picture is a motion vector. The motion vector may point to integer, half- or quarter-sample positions in the luma component of the reference picture. Half- or quarter-sample positions are generated by interpolating the samples of the reference picture. Each motion vector is differentially coded from the motion vectors of neighbouring blocks.

The prediction block may be generated from a single prediction region in a reference picture, for a P or B macroblock, or from two prediction regions in reference pictures, for a B macroblock [i]. Optionally, the prediction block may be weighted according to the temporal distance between the current and reference picture(s), known as weighted prediction. In a B macroblock, a block may be predicted in direct mode, in which case no residual samples or motion vectors are sent and the decoder infers the motion vector from previously received vectors.

Example

A macroblock in frame *n* is shown in Figure 6.17. The macroblock is divided into two partitions, each consisting of 8 × 16 luma samples and corresponding chroma samples. The left partition (A)

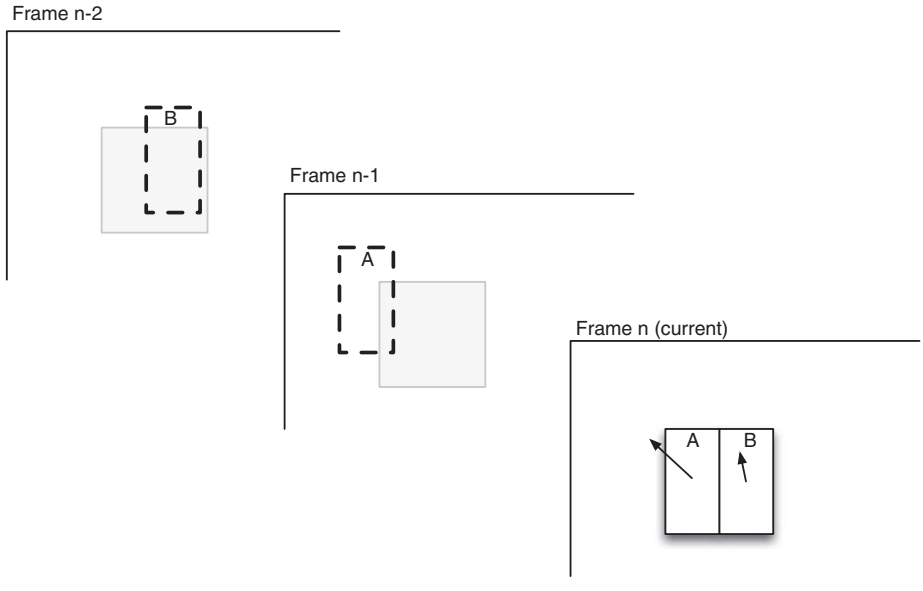


Figure 6.17 P macroblock prediction example

is predicted from a region in the previous frame, frame *n−1*, and the right partition (B) is predicted from a region in frame *n−2*. Partition A has a motion vector (−6.5, −5.75), i.e. the reference region is offset by −6.5 samples in the x direction (left) and −5.75 samples in the y direction (up). Partition B has motion vector (−1.25, −4), i.e. −1.25 samples in the x direction (left) and −4 samples in the y direction (up).

To summarize the process of coding an inter-predicted macroblock (note that the steps need not occur in this exact order):

1. Interpolate the picture(s) in the Decoded Picture Buffer to generate $^1/_4$-sample positions in the luma component and 1/8-sample positions in the chroma components. (section 6.4.2).
2. Choose an inter prediction mode from the following options:
   (a) Choice of reference picture(s), previously-coded pictures available as sources for prediction. (section 6.4.1).
   (b) Choice of macroblock partitions and sub-macroblock partitions, i.e. prediction block sizes. (section 6.4.3).

(c) Choice of prediction types:
   (i) prediction from one reference picture in list 0 for P or B macroblocks or list 1 for B macroblocks only (section 6.4.5.1).
   (ii) bi-prediction from two reference pictures, one in list 0 and one in list 1, B macroblocks only, optionally using weighted prediction (section 6.4.5.2).

3. Choose motion vector(s) for each macroblock partition or sub-macroblock partition, one or two vectors depending on whether one or two reference pictures are used.
4. Predict the motion vector(s) from previously-transmitted vector(s) and generate motion vector difference(s). Optionally, use Direct Mode prediction, B macroblocks only. (section 6.4.4).
5. Code the macroblock type, choice of prediction reference(s), motion vector difference(s) and residual. (Chapters 5 and 7).
6. Apply a deblocking filter prior to storing the reconstructed picture as a prediction reference for further coded pictures. (section 6.5).

6.4.1 Reference pictures

Inter prediction makes use of previously coded pictures that are available to the decoder. Slices are received, decoded to produce pictures and displayed. They are also stored in the Decoded Picture Buffer (DPB), in which case they may be used as reference pictures for inter prediction. See Chapter 5 for a more detailed discussion of the DPB and reference picture lists.

The pictures in the DPB are indexed, i.e. listed in a particular order, in the following Lists, depending on whether the current MB is in a P or a B slice.

List0 (P slice): A single list of all the reference pictures. By default, the first picture in the List is the most recently decoded picture.

List0 (B slice): A list of all the reference pictures. By default, the first picture in the List is the picture **before** the current picture in display order.

List1 (B slice): A list of all the reference pictures. By default, the first picture in the List is the picture **after** the current picture in display order.

Hence the construction of List 0 is different depending on whether the current MB occurs in a P or B slice. A P or B macroblock uses reference pictures in one or two of these Lists to form a macroblock prediction (Table 6.3).

Table 6.3 Reference picture sources

Slice type	MB type	Reference picture sources
P	P	List0 (P slice)
B	P	List0 (B slice)
B	B	List0 (B slice) and List1 (B slice)

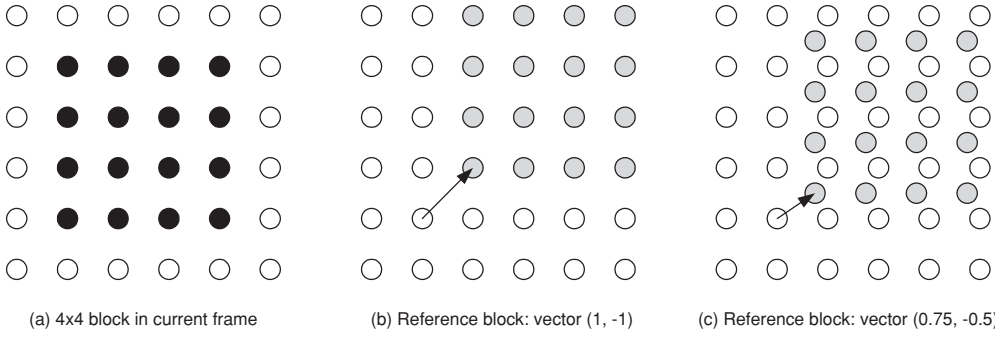


Figure 6.18 Example of integer and sub-pixel prediction

6.4.2 *Interpolating reference pictures*

Each partition in an inter-coded macroblock is predicted from an area of the same size in a reference picture. The offset between the two areas, the motion vector, has $^1/_4$-pixel resolution for the luma component and 1/8-pixel resolution for the chroma components. The luma and chroma samples at sub-pixel positions do not exist in the reference picture and so it is necessary to create them using interpolation from nearby image samples [ii]. In Figure 6.18, a 4×4 block in the current frame (a) is predicted from a neighbouring region of the reference picture. If the horizontal and vertical components of the motion vector are integers (b), the relevant samples in the reference block actually exist, shown as grey dots. If one or both vector components are fractional values (c), the prediction samples, shown as grey dots, are generated by interpolation between adjacent samples in the reference frame, shown as white dots.

Example

Figure 6.19 shows a small region of the current frame, containing a vertical object on a white background. We want to find a match for the 4×4 block shown in Figure 6.20, marked with small stars. The corresponding region of the reference frame is shown in Figure 6.21. Note that the vertical object is **not** aligned with the sample positions in Figure 6.21; the object has moved by a non-integer number of pixels between frames. Without any interpolation, it is not possible to find a good match in the reference region. The best match will be something like the one shown in Figure 6.22. We may be able to do better by interpolating between the samples of the reference frame to generate half-pixel positions (Figure 6.23). Searching the interpolated reference frame gives a better match (Figure 6.24), best match indicated by stars. The match is not perfect – the luminance levels are not quite the same as those of the original 4×4 block (Figure 6.20) – but the prediction is better than the integer-pixel match. A better prediction gives a smaller residual and hence better compression. In general, 'finer' interpolation, i.e. increasing the number of interpolation stages, reduces the residual, at the expense of increased computation and more bits required to send motion vectors.

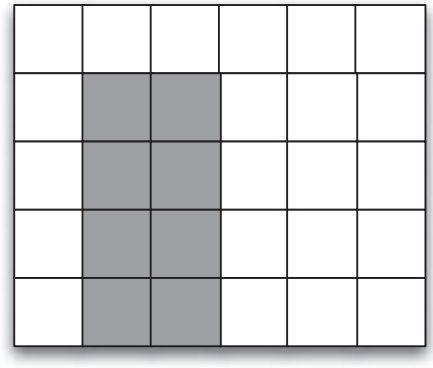


Figure 6.19 Current region

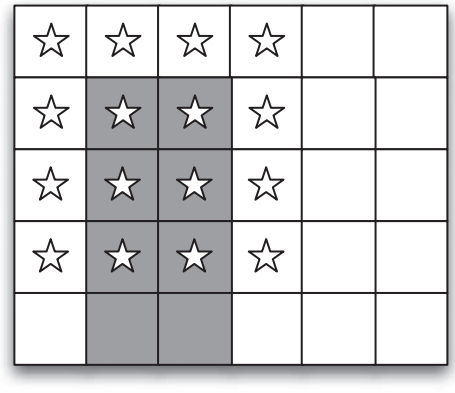


Figure 6.20 4×4 block to be predicted

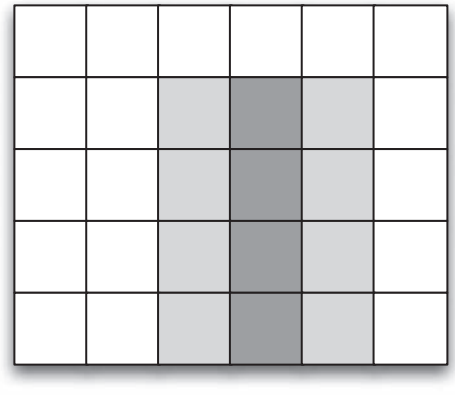


Figure 6.21 Reference region

Figure 6.22 Prediction from integer samples

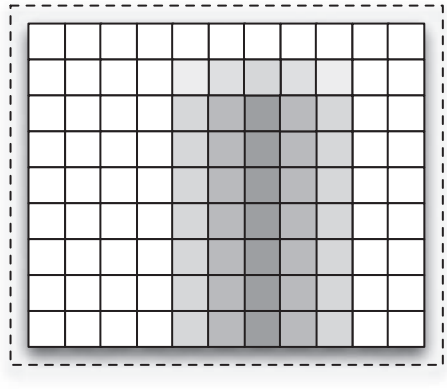


Figure 6.23 Reference region, half-pixel interpolated

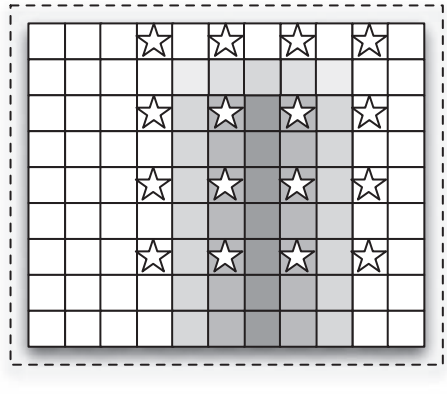


Figure 6.24 Prediction from interpolated samples

6.4.2.1 Generating interpolated sub-pixel samples

Luma component

The half-pel samples in the luma component of the reference picture are generated first, Figure 6.25, grey markers. Each half-pel sample that is adjacent to two integer samples, e.g. b, h, m, s in Figure 6.25, is interpolated from integer-pel samples using a 6 tap Finite Impulse Response (FIR) filter with weights (1/32, −5/32, 5/8, 5/8, −5/32, 1/32). For example, half-pel sample **b** is calculated from the 6 horizontal integer samples E, F, G, H, I and J using a process equivalent to:

$$\mathbf{b} = \text{round}((\text{E} - 5\text{F} + 20\text{G} + 20\text{H} - 5\text{I} + \text{J})/32)$$

Similarly, **h** is interpolated by filtering A, C, G, M, R and T. Once all of the samples adjacent to integer samples have been calculated, the remaining half-pel positions are calculated by interpolating between six horizontal or vertical half-pel samples from the first set of operations. For example, **j** is generated by filtering cc, dd, h, m, ee and ff. Note that the result is the same whether j is interpolated horizontally or vertically. The 6-tap interpolation filter is relatively complex but produces an accurate fit to the integer-sample data and hence good motion compensation performance.

Once all the half-pixel samples are available, the quarter-pixel positions are produced by linear interpolation Figure 6.26. Quarter-pixel positions with two horizontally or vertically adjacent half- or integer-pixel samples, e.g. **a**, **c**, **i**, **k** and **d**, **f**, **n**, **q** in Figure 6.26, are linearly interpolated between these adjacent samples, for example:

$$\mathbf{a} = \text{round}((\text{G} + \text{b})/2)$$

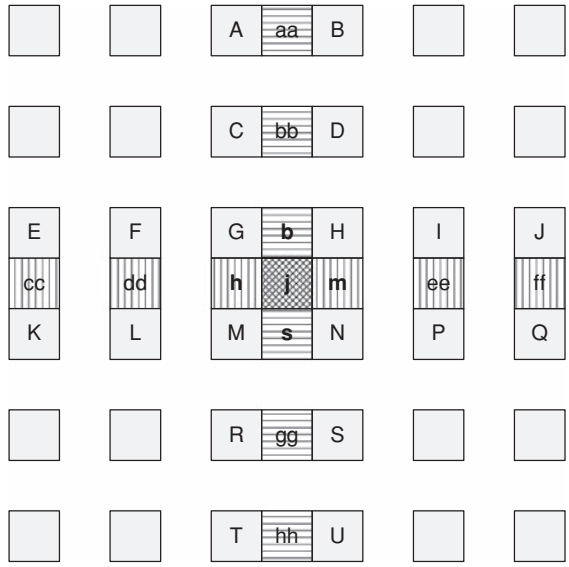


Figure 6.25 Interpolation of luma half-pel positions

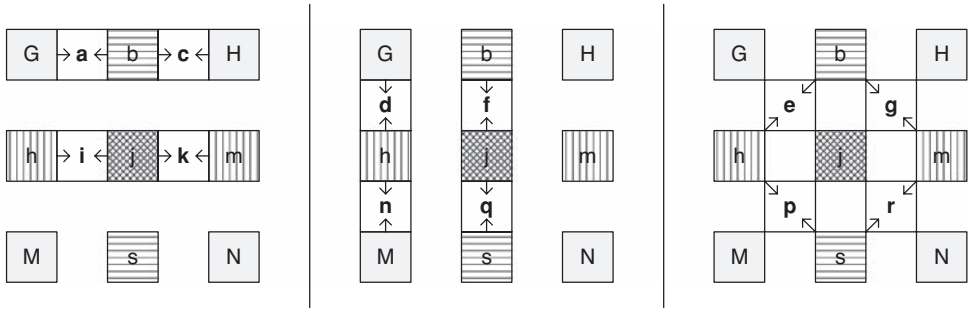


Figure 6.26 Interpolation of luma quarter-pel positions

The remaining quarter-pixel positions, **e**, **g**, **p** and **r** in the figure, are linearly interpolated between a pair of diagonally opposite half-pixel samples. For example, **e** is interpolated between b and h.

Figure 6.27 shows the result of interpolating the reference region shown in Figure 3.16 with quarter-pixel resolution.

Chroma components

Quarter-pel resolution motion vectors in the luma component require eighth-pel resolution vectors in the chroma components, assuming 4:2:0 sampling. Interpolated samples are

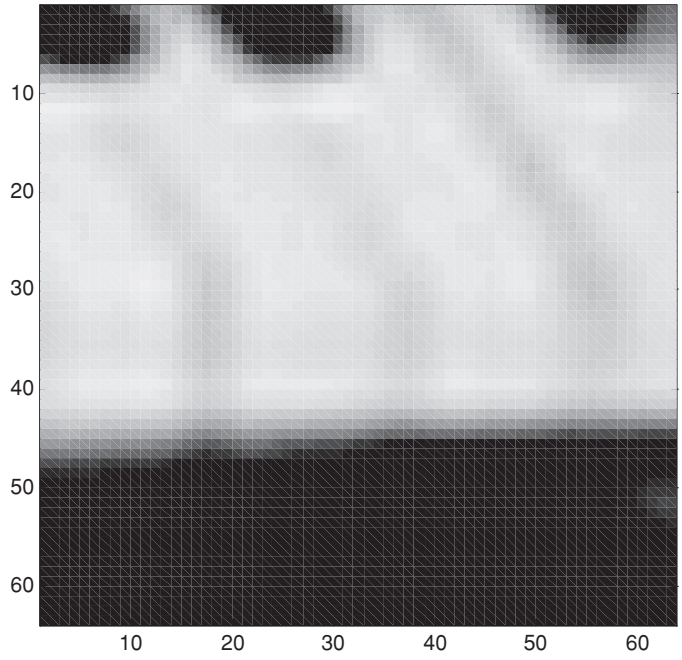


Figure 6.27 Luma region interpolated to quarter-pel positions

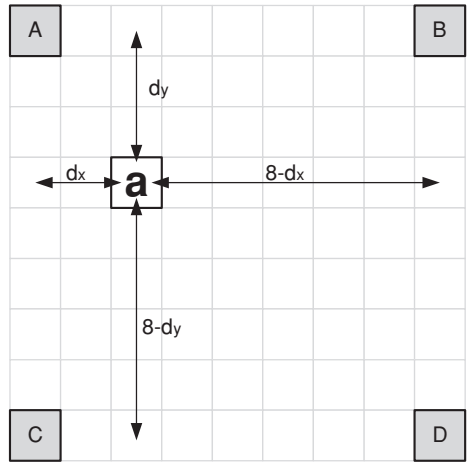


Figure 6.28 Interpolation of chroma eighth-pel positions

generated at 1/8-pel intervals between integer samples in each chroma component using linear interpolation (Figure 6.28). Each sub-pel position **a** is a linear combination of the neighbouring integer pel positions A, B, C and D:

$$\mathbf{a} = \text{round}([(8 - d_x).(8 - d_y)A + d_x.(8 - d_y)B + (8 - d_x).d_yC + d_x.d_yD]/64)$$

In Figure 6.28, d_x is 2 and d_y is 3, so that:

$$a = \text{round}([30A + 10B + 18C + 6D]/64)$$

6.4.3 Macroblock partitions

Each 16 × 16 P or B macroblock may be predicted using a range of block sizes. The macroblock is split into one, two or four **macroblock partitions**: either

(a) one 16 × 16 macroblock partition (covering the whole MB),
(b) two 8 × 16 partitions,
(c) two 16 × 8 partitions or
(d) four 8 × 8 partitions.

If 8 × 8 partition size is chosen, then each 8 × 8 block of luma samples and associated chroma samples, a **sub-macroblock**, is split into one, two or four **sub-macroblock partitions**): one 8 × 8, two 4 × 8, two 8 × 4 or four 4 × 4 sub-MB partitions (Figure 6.29).

Each macroblock partition and sub-macroblock partition has one or two motion vectors (x, y), each pointing to an area of the same size in a reference frame that is used to predict the current partition. A partition in a P macroblock has one reference frame and one motion vector; a partition in a B macroblock has one or two reference frames and one or two corresponding motion vectors. Each macroblock partition may be predicted from different reference frame(s).

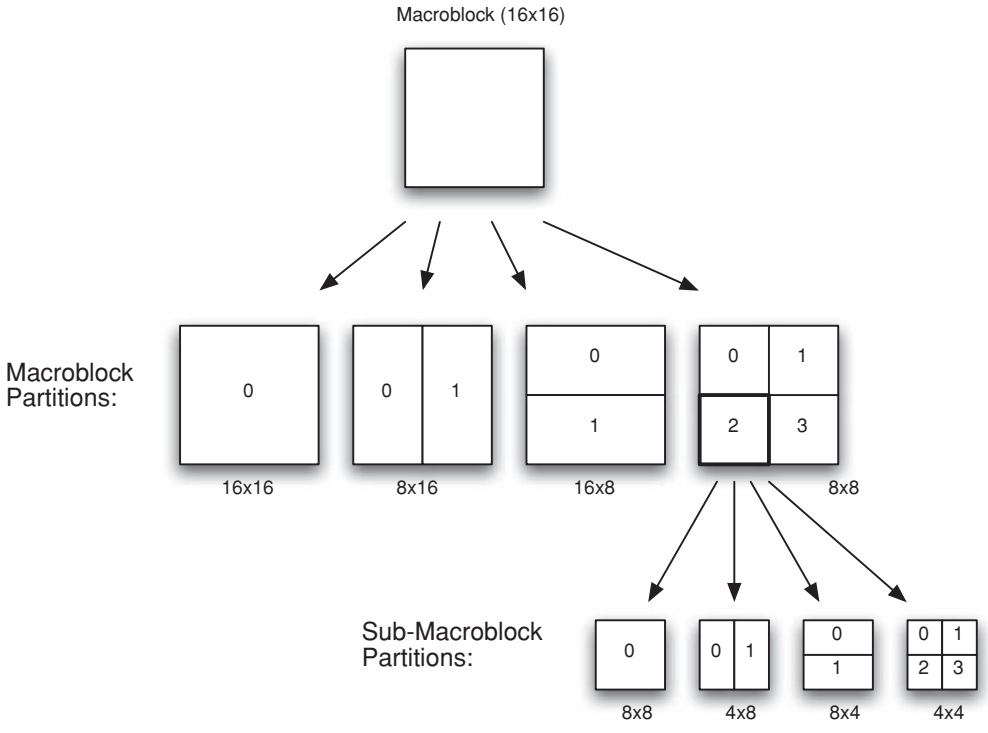


Figure 6.29 Macroblock partitions and sub-macroblock partitions

However, the sub-macroblock partitions within an 8 × 8 sub-macroblock share the same reference frame(s). Table 6.4 summarizes the information sent with each macroblock.

6.4.4 Motion vector prediction

Encoding a motion vector for each partition can cost a significant number of bits, especially if small partition sizes are chosen. Motion vectors for neighbouring partitions are often highly correlated and so each motion vector is predicted from vectors of nearby, previously coded partitions. A predicted vector, MVp, is formed based on previously calculated motion vectors and MVD, the difference between the current vector and the predicted vector, is encoded and

Table 6.4 Reference frames and motion vectors for P and B macroblocks

Macroblock type:	P	B
Reference frame indices:	One per macroblock partition, from list 0	One or two per macroblock partition, from list 0, list 1 or both lists.
Motion vector pairs (x,y):	One per macroblock partition or sub-macroblock partition	One or two per macroblock or sub-macroblock partition

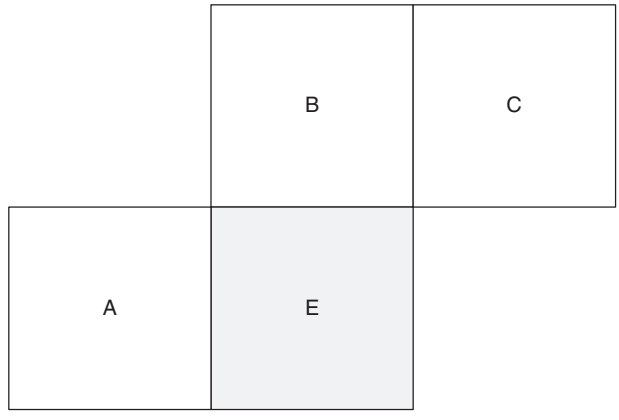


Figure 6.30 Current and neighbouring partitions : same partition sizes

transmitted. The method of forming the prediction MVp depends on the motion compensation partition size and on the availability of nearby vectors.

Let E be the current macroblock, macroblock partition or sub-macroblock partition, let A be the partition or sub-macroblock partition immediately to the left of E, let B be the partition or sub-macroblock partition immediately above E and let C be the partition or sub-macroblock partition above and to the right of E. If there is more than one partition immediately to the left of E, the topmost of these partitions is chosen as A. If there is more than one partition immediately above E, the leftmost of these is chosen as B. Figure 6.30 illustrates the choice of neighbouring partitions when all the partitions have the same size, 16 × 16 in this case, and Figure 6.31 shows an example of the choice of prediction partitions when the neighbouring partitions have different sizes from the current partition E.

1. For transmitted partitions excluding 16 × 8 and 8 × 16 partition sizes, MVp is the median of the motion vectors for partitions A, B and C.
2. For 16 × 8 partitions, MVp for the upper 16 × 8 partition is predicted from B and MVp for the lower 16 × 8 partition is predicted from A.

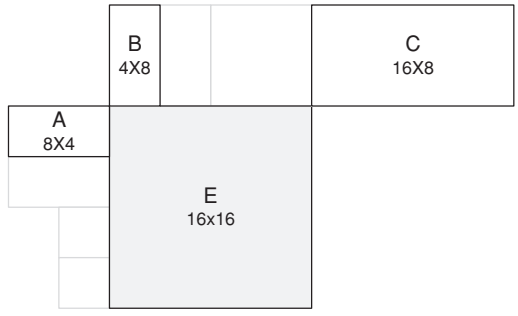


Figure 6.31 Current and neighbouring partitions : different partition sizes

3. For 8 × 16 partitions, MVp for the left 8 × 16 partition is predicted from A and MVp for the right 8 × 16 partition is predicted from C.
4. For skipped macroblocks, a 16 × 16 vector MVp is generated as in case (1) above, as if the block were encoded in 16 × 16 Inter mode.

If one or more of the previously transmitted blocks shown in the figure is not available, e.g. if it is outside the current picture or slice, the choice of MVp is modified accordingly. At the decoder, the predicted vector MVp is formed in the same way and added to the decoded vector difference MVD. In the case of a skipped macroblock, there is no decoded vector and a motion-compensated macroblock is produced using MVp as the motion vector.

6.4.4.1 Bipredicted macroblock motion vector prediction

A bipredicted macroblock in a B slice has two motion vectors. If one reference frame is a past frame and the other is a future frame, each of the two vectors is predicted from neighbouring motion vectors that have the **same temporal direction**, i.e. a vector for the current macroblock pointing to a past frame is predicted from other neighbouring vectors that also point to past frames.

If both reference frames are past frames, relative to the current frame, or both are future frames, then one of the vectors is encoded as a **scaled motion vector**. The vector pointing to the list 0 reference (MV0) is encoded as described above and a motion vector is calculated by scaling MV0 based on the temporal distances between the current frame, the list 0 frame and the list 1 frame. The list 1 vector (MV1) is then encoded differentially from the scaled motion vector.

Example

Current frame number 86 is predicted from list 0 frame 85 and list 1 frame 83. The list 0 vector (Figure 6.32a) is calculated as (+1, −1) and the list 1 vector (Figure 6.32b) is calculated as (+1.5, −2.5).

The temporal distance to the list 0 frame is 1 and the distance to the list 1 frame is 3. The list 0 vector is therefore scaled by (3/1 = 3) to produce a scaled motion vector (smv) of (+3, −3) (Figure 6.32b). The difference between the actual list 1 vector and smv is (−1.5, +0.5) (Figure 6.32c) and this is encoded and transmitted.

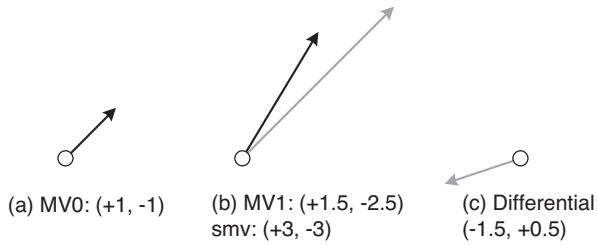


Figure 6.32 Scaled motion vector example

6.4.4.2 Direct mode motion vector prediction

No motion vector is transmitted for a B slice macroblock or partition encoded in **direct mode**. Instead, the decoder calculates list 0 and list 1 vectors based on previously coded vectors and uses these to carry out bipredicted motion compensation of the decoded residual samples (section 6.4.5).

A flag in the slice header indicates whether a spatial or temporal method shall be used to calculate the vectors for direct mode macroblocks or partitions. In **spatial direct** mode, list 0 and list 1 vectors of neighbouring previously coded macroblocks or partitions in the same slice are used to calculate the list 0 and list 1 vectors of the current MB or partition. In **temporal direct** mode, the decoder carries out the following steps:

1. Find the list 0 reference frame for the co-located MB or partition in the list 1 frame. This list 0 reference becomes the list 0 reference of the current MB or partition.
2. Find the list 0 vector, MV, for the co-located MB or partition in the list 1 frame.
3. Scale vector MV based on the temporal distance between the current and list 1 frames: this is the new list 1 vector MV1.
4. Scale vector MV based on the temporal distance between the current and list 0 frames: this is the new list 0 vector MV0.

Example: Temporal Direct mode

The list 1 reference for the current macroblock occurs two frames after the current frame (Figure 6.33). The co-located MB in the list 1 reference has a vector MV(+2.5, +5) pointing to a list 0 reference frame that occurs three frames before the current frame. The decoder calculates MV1(−1, −2) and MV0(+1.5, +3) pointing to the list 1 and list 0 frames respectively. These vectors are derived from MV and have magnitudes proportional to the temporal distance to the list 0 and list 1 reference frames.

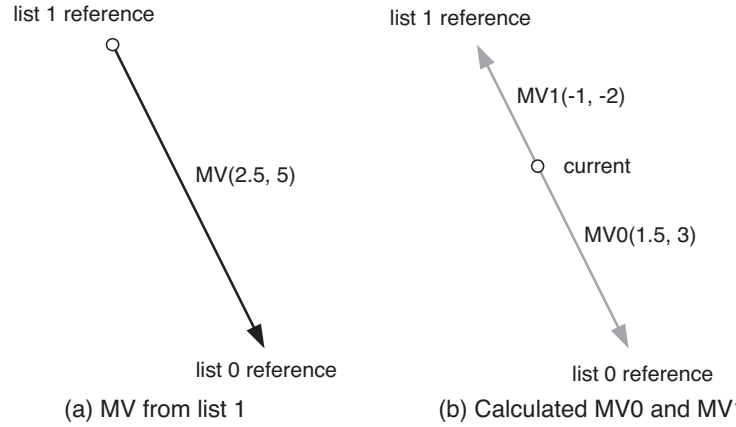


Figure 6.33 Temporal direct motion vector example

6.4.5 Motion compensated prediction

An H.264 encoder or decoder creates a motion-compensated prediction from reference picture(s) in the DPB using the motion vectors and reference list indices signalled in a P or B macroblock (Figure 6.34). Reference list indices identify the reference picture(s) used to form the prediction; motion vectors indicate the offset from the current macroblock or partition to corresponding reference area(s) in the reference picture(s).

Each macroblock partition in a P macroblock is predicted from one reference, from list0. Each macroblock partition in a B macroblock is predicted from one reference from list0 or list1 or bi-predicted from two references, one from list0, one from list1.

6.4.5.1 One reference

A motion vector is signalled for each macroblock partition or sub-macroblock partition. The vector is an offset (x, y) to a region in a reference picture of the same size as the current macroblock partition or sub-macroblock partition. This region forms the prediction for the current partition.

6.4.5.2 Two references : biprediction

Two motion vectors are signalled for the macroblock partition or sub-macroblock partition, each pointing to a region of the same size in a reference picture, one from list0, one from list1.

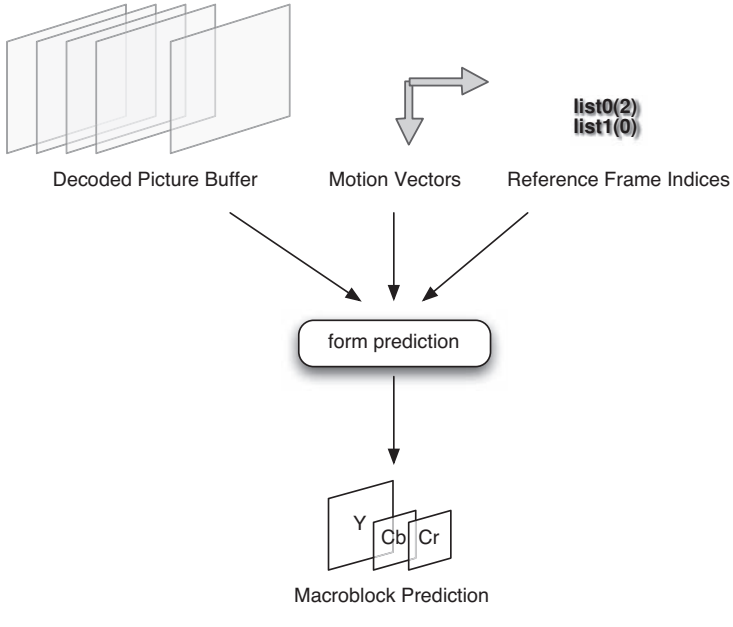


Figure 6.34 Forming a motion compensated prediction

Each sample of the prediction is calculated as an average of the samples in the list0 and list1 regions (biprediction).

Example

A macroblock is predicted in B_Bi_16 × 16 mode, i.e. a single bipredicted 16 × 16 region. The reference areas indicated by the list0 and list1 vectors and the 16 × 16 prediction block are shown in Figure 6.35.

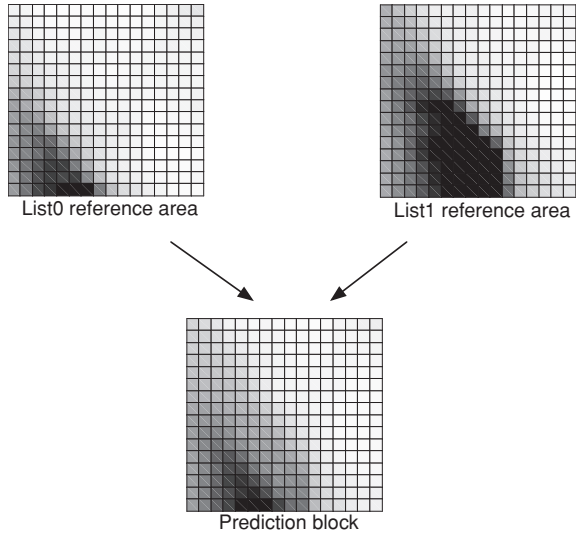


Figure 6.35 Biprediction example

6.4.5.3 Weighted prediction

Weighted prediction is a method of scaling the samples of motion-compensated prediction data in a P or B slice macroblock. There are three types of weighted prediction in H.264:

(a) P slice macroblock, 'explicit' weighted prediction
(b) B slice macroblock, 'explicit' weighted prediction
(c) B slice macroblock, 'implicit' weighted prediction.

Each prediction sample pred0(i,j) or pred1(i,j) is scaled by a weighting factor W_0 or W_1 prior to motion-compensated prediction. In the 'explicit' types, the weighting factor(s) are determined by the encoder and transmitted in the slice header. If 'implicit' prediction is used, W_0 and W_1 are calculated based on the relative temporal positions of the list 0 and list 1 reference frames. A larger weighting factor is applied if the reference frame is temporally close to the current frame and a smaller factor is applied if the reference frame is temporally further away from the current frame.

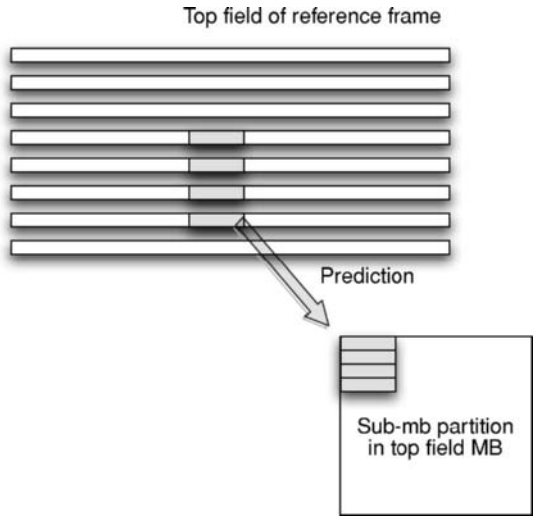


Figure 6.36 MBAFF: prediction from corresponding field

One application of weighted prediction is to allow explicit or implicit control of the relative contributions of reference frames to the motion-compensated prediction process. For example, weighted prediction may be particularly effective in coding of 'fade' transitions where one scene fades into another.

6.4.5.4 Frame / field prediction

Chapter 5 described how an H.264 encoder can choose between Frame and Field coding at the picture level, i.e. coding the complete picture as a frame or as a field, and/or at the macroblock level, coding a vertically adjacent pair of macroblocks as two frame macroblocks or two field macroblocks, Macroblock Adaptive Frame Field Coding or MBAFF.

Prediction of a Frame picture proceeds as described in the previous sections. Each MB is predicted from previously-coded **frame(s)** stored in the DPB. Prediction of a Field picture proceeds in essentially the same way, except that each MB is predicted from previously-coded **field(s)** in the DPB. As discussed in Chapter 5, the stored pictures in the DPB are organized as a series of frames or fields depending on the mode of the current picture, whether frame or field coding.

If MBAFF is enabled, prediction proceeds as follows. If the current MB is a frame MB, each partition is predicted from a region in a reference frame. If the current MB is a field MB, each partition is predicted from a region in the corresponding field, top or bottom, in a reference frame. In the example in Figure 6.36, a 4×4 sub macroblock partition in the top field MB of a macroblock pair is predicted from a 4×4 region in the top field of the chosen reference frame.

6.4.6 Inter prediction examples

H.264/AVC offers many possible options for inter prediction. Selection of appropriate inter prediction modes has a significant influence on the compression efficiency of an H.264 codec. The following examples will help illustrate the factors that affect inter prediction choices.

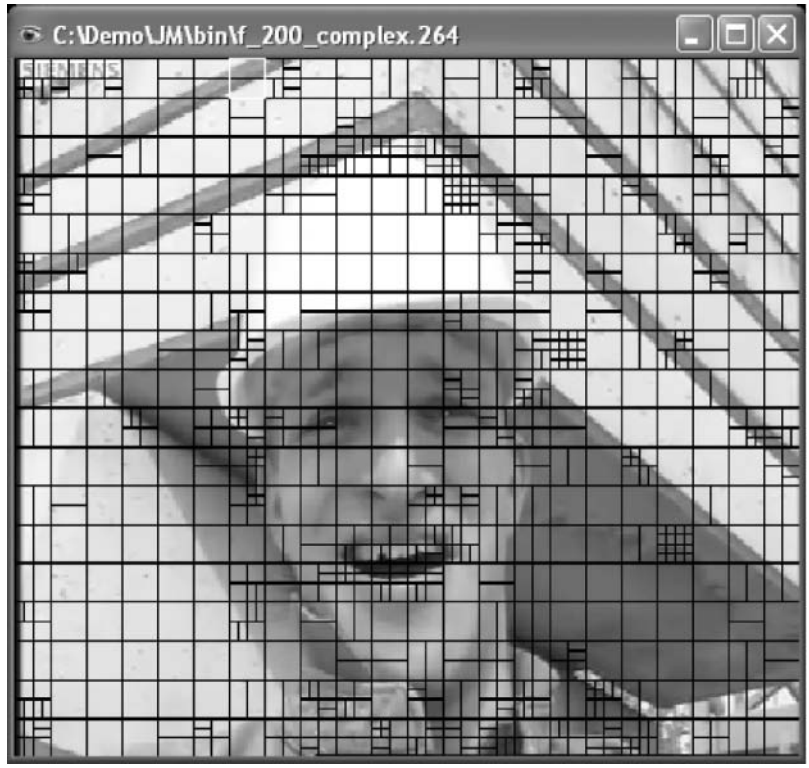


Figure 6.37 P slice showing partition choices. Reproduced by permission of Elecard

Motion compensated prediction tends to be more accurate when small partition sizes are used, especially when motion is relatively complex. However, more partitions in a macroblock mean that more bits must be sent to indicate the motion vectors and choice of partitions. Typically, an encoder will choose larger partitions in homogeneous areas of a frame with less texture / movement and smaller partitions in areas of more complex motion. Figure 6.37 shows a frame coded as a P slice, with the choice of partitions overlaid on the frame. Many macroblocks are coded using a single 16 × 16 partition and hence a single motion vector. Around complicated areas of movement such as the mouth, smaller partitions and more vectors are chosen.

In a typical B slice, some of the macroblocks can be skipped, B-skip mode, i.e. the macroblock is reconstructed using motion compensated prediction from two reference frames, with no motion vectors or residual data being sent. An example of a B slice is shown in Figure 6.38. This is a B picture from a 30 frames per second CIF-resolution sequence coded at 200kbits/second. The light circles indicate macroblocks coded in B-skip mode. It is clear that most of the macroblocks are skipped. Only a few macroblocks – typically around edges or features of moving objects such as the face – contain transmitted motion vectors and/or residual data.

Making the 'best' choice of motion compensation partitions can have a significant impact on the compression efficiency of an H.264 sequence. The trade-off between partition choices

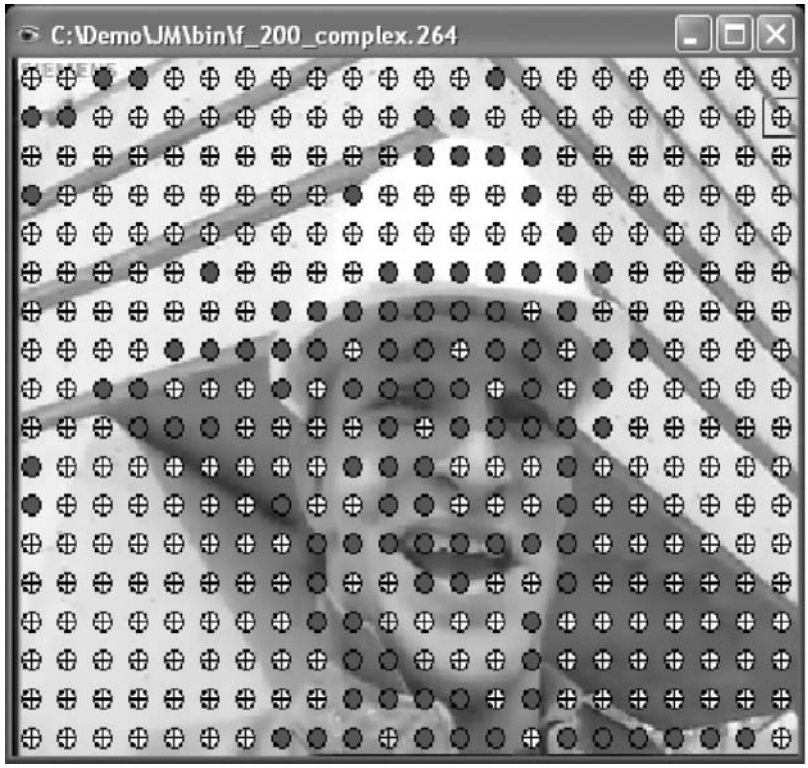


Figure 6.38 B slice showing macroblock modes. Light-shaded circles are skipped macroblocks. Reproduced by permission of Elecard.

and residual compression can be optimized using Rate Distortion Optimized (RDO) mode selection (Chapter 9).

Example 1: P slice

Figure 6.39 shows an example of the inter prediction of a macroblock in a P slice. The macroblock in frame n uses the following combination of 8 × 8 partitions and sub-macroblock partitions:

Partition 0 One 8 × 8 partition Prediction reference: List0(2) Motion vector: (0,0)	Partition 1 One 8 × 8 partition Prediction reference: List0(1) Motion vector: (+9.5, +3.75)
Partition 2 Two 8 × 4 sub-macroblock partitions Prediction reference: List0(0) Motion vectors: (−7.25, 0) and (0,0)	Partition 3 Two 8 × 4 sub-macroblock partitions Prediction reference: List0(0) Motion vectors: (5.5, −2) and (0,0)

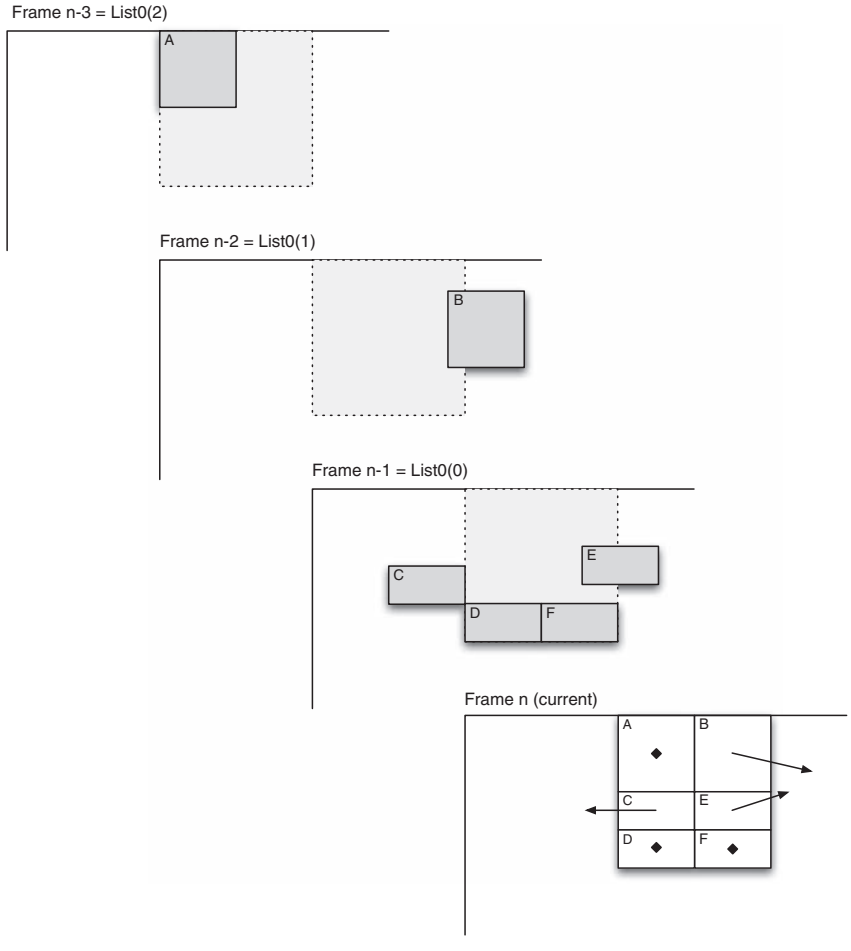


Figure 6.39 Inter prediction example, P slice

List0(0) points to the previous frame in display order, frame n−1, List0(1) is frame n−2 and List0(2) is frame n−3. This is the default order for P-slice reference frames, i.e. frames are referenced in increasing temporal distance from the current frame (Chapter 5). Note that all sub-macroblocks in each 8 × 8 partition use the same reference frame (section 6.4.3). Figure 6.39 shows the corresponding regions in the reference frames, shown as dark grey areas, used to form predictions of each block in the current macroblock.

Example 2: B slice

In a B macroblock (Figure 6.40) each partition may have one or two prediction references. In this example, a macroblock in the current frame (n) has two 8 × 16 partitions:

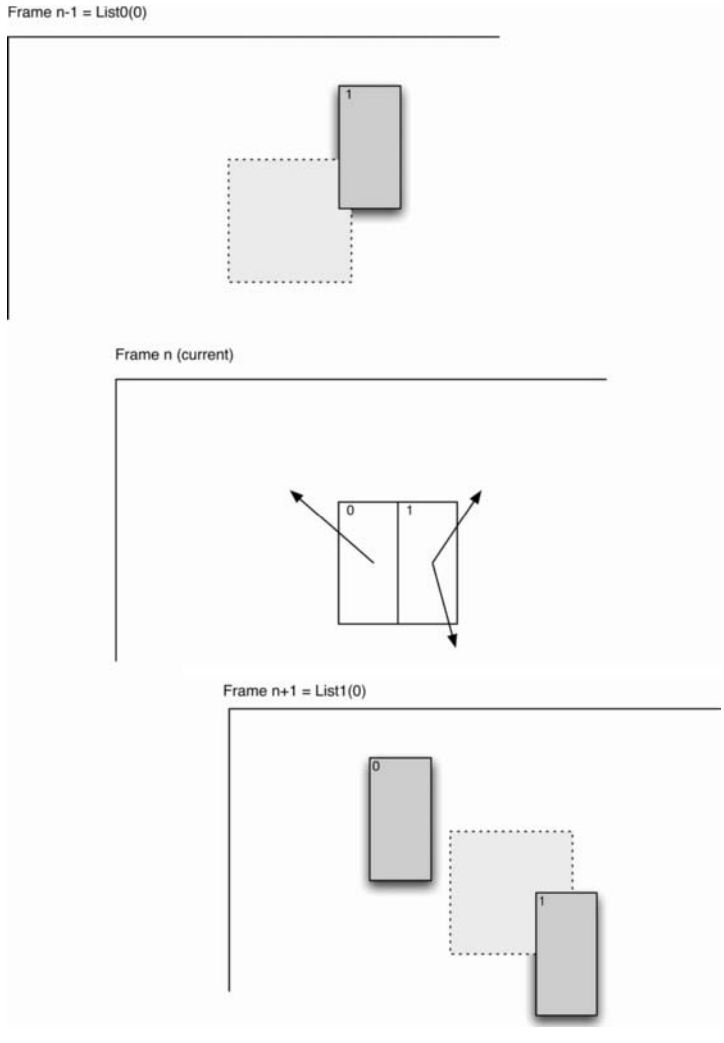


Figure 6.40 Inter prediction example, B slice

Partition 0
Prediction reference: List1(0), which points to frame n + 1, the next frame in display order.
Motion vector: (−10, −9.5)

Partition 1
Prediction references: List0(0), which points to frame n−1, the previous frame in display order, and List1(0), which points to frame n + 1.
Motion vectors: (+7.25, −9) (List0) and (+6, +8.25) (List1).

Hence Partition 0 is predicted from a corresponding area in frame (n + 1) whereas Partition 1 is bipredicted or interpolated between areas in frame (n−1) and frame (n + 1).

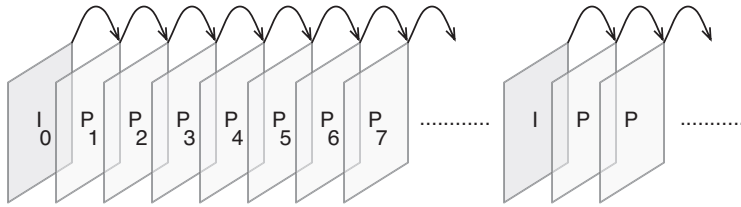


Figure 6.41 Low delay prediction structure

6.4.7 Prediction structures

H.264 offers many different options for choosing reference pictures for inter prediction. An encoder will typically use reference pictures in a structured way. Some examples of reference picture structures are presented here.

6.4.7.1 Low delay, minimal storage

The prediction structure shown in Figure 6.41 uses only I and P slices. It is compatible with the Baseline Profile or Constrained Baseline Profile of H.264, which do not allow B slices, and would be suitable for an application requiring low delay and/or minimal storage memory at the decoder. The first frame is coded as an I slice and subsequent frames are coded as P slices, predicted from the previous frame. Prediction efficiency is relatively low, because only one prediction direction and one reference is allowed for each frame. However, the decoder can display each frame as soon as it is decoded, minimising delay, and the decoder only needs to store one reference frame, minimising storage requirements. This type of structure would be suitable for applications such as videoconferencing where latency must be kept to a minimum. Note that I slices may be inserted in the stream at intervals to limit the propagation of transmission errors and to enable random access to the coded sequence.

6.4.7.2 'Classic' Group of Pictures structure

Earlier standards such as MPEG-1 and MPEG-2 supported the Group of Pictures structure shown in Figure 6.42. The Group of Pictures (GOP) starts with an I slice. P slices are inserted at intervals, with B slices between the I and P slices. The I and P slices are used for reference;

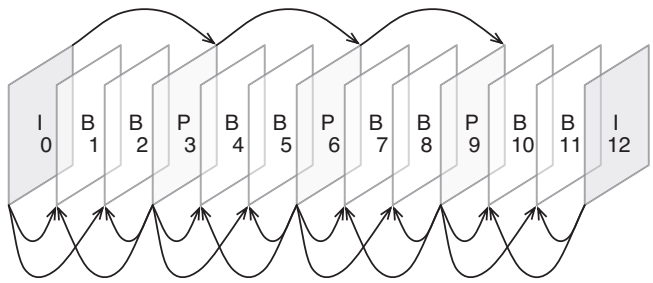


Figure 6.42 'Classic' Group of Pictures prediction structure

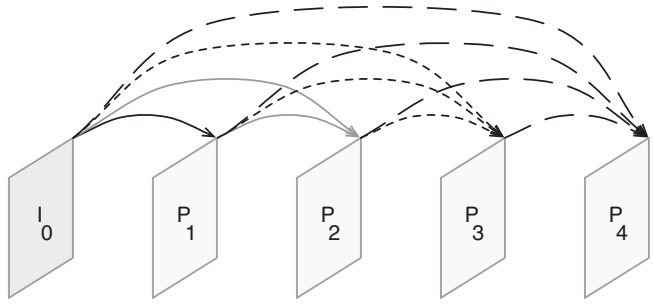


Figure 6.43 IPPP . . . with multiple reference pictures

the B slices are not used for reference, i.e. all pictures are predicted from I or P slices. Each P slice is predicted from the preceding I or P slice and each B slice is predicted from the I and/or P slices on either side of it.

This structure provides higher compression efficiency than the structure shown in Figure 6.41 because prediction in a B slice tends to be more efficient than in a P slice due to the availability of two reference pictures. The disadvantage is increased delay and larger frame storage requirements. The decoder must receive slices I0 and P3 before slices 1 and 2 can be decoded, i.e. the delay is a minimum of three frames in this example. Slices 0 and 3 must be stored by the decoder because they are reference frames for slices 1 and 2.

6.4.7.3 Multiple reference frames

H.264 makes it possible to create other prediction structures, for example using multiple reference frames for prediction. Figure 6.43 shows an IPPP . . . prediction structure in which all the previously coded slices are available as reference frames. Slice P1 is predicted from slice I0; slice P2 is predicted from slices I0 and P1; slice P3 is predicted from slices I0, P1 and P2; and so on. This means that the encoder can search up to N reference frames to find the best match for each P macroblock. This can improve compression efficiency at the expense of (a) increased computational expense at the encoder and (b) increased storage at encoder and decoder, since N reference frames must be stored.

6.4.7.4 Hierarchical prediction structures

Figure 6.44 shows an example of a hierarchical GOP structure. This type of structure is made possible by H.264's flexible prediction options. The GOP starts and finishes with I slices 0 and 12. Next, slice B6 is predicted using I0 and I12 as references; note that B6 is halfway between I0 and I12 in display order. B3 is predicted from I0 and B6; B9 is predicted from B6 and I12. B1 and B2 are predicted from I0 and B3 and so on. The GOP is composed of a series of layers, a dyadic or pyramid decomposition, see Table 6.5. This type of structure can give improved compression performance [iii] if the quantization parameters of the layers are carefully controlled such that the QP increases as the layer number increases. Hierarchical or pyramid GOP structures were first proposed as a method of providing temporal scalability (Chapter 10) but they may be used in any H.264 codec that supports B slices.

Chapter 9 compares the compression performance of several prediction structures.

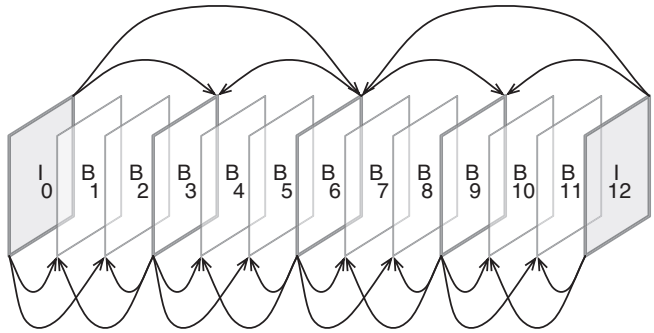


Figure 6.44 Hierarchical GOP structure

6.5 Loop filter

A filter is applied to every decoded macroblock to reduce blocking distortion [iv]. The deblocking filter is applied after the inverse transform in the encoder before reconstructing and storing the macroblock for future predictions and in the decoder before reconstructing and displaying the macroblock. The filter smooths block edges, improving the appearance of decoded frames. The filtered image is used for motion-compensated of future frames and this generally improves compression performance because the filtered image is a more faithful reproduction of the original frame than a blocky, unfiltered image.[1]

Filtering is applied to vertical or horizontal edges of 4×4 blocks in a macroblock excluding edges on slice boundaries, in the following order:

1. Filter 4 vertical boundaries of the luma component in order a,b,c,d in Figure 6.45
2. Filter 4 horizontal boundaries of the luma component in order e,f,g,h, Figure 6.45
3. Filter 2 vertical boundaries of each chroma component (i,j)
4. Filter 2 horizontal boundaries of each chroma component (k,l)

Each filtering operation affects up to **three** pixels on either side of the boundary. Figure 6.46 shows four pixels on either side of a vertical or horizontal boundary in adjacent blocks p and q, p0,p1,p2,p3 and q0,q1,q2,q3. The 'strength' of the filter, i.e. the amount

Table 6.5 Layers in hierarchical GOP example

Layer	Slices
0	I0, I12
1	B6
2	B3, B9
3	B1, B2, B4, B5, B7, B8, B10, B11

[1] Intra-coded macroblocks are filtered, but intra prediction (section 6.3) is carried out using **unfiltered** reconstructed macroblocks to form the prediction.

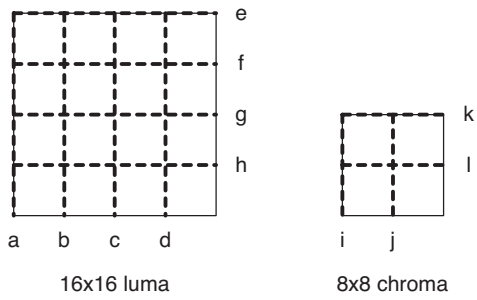


Figure 6.45 Edge filtering order in a macroblock

of filtering, depends on the current quantizer parameter QP, the coding modes of neighbouring blocks and the gradient of image samples across the boundary.

6.5.1 Boundary strength

The choice of filtering outcome depends on the **boundary strength** and on the **gradient** of image samples across the boundary. The boundary strength parameter **Bs** is chosen according to the following rules:

p or q is intra coded **and** boundary is a macroblock boundary	Bs = 4, strongest filtering
p or q is intra coded and boundary is **not** a macroblock boundary	Bs = 3
neither p or q is intra coded; p or q contain coded coefficients	Bs = 2
neither p or q is intra coded; neither p or q contain coded coefficients; p and q have different reference frames **or** a different number of reference frames **or** different motion vector values	Bs = 1
neither p or q is intra coded; neither p or q contain coded coefficients; p and q have same reference frame and identical motion vectors	Bs = 0, no filtering

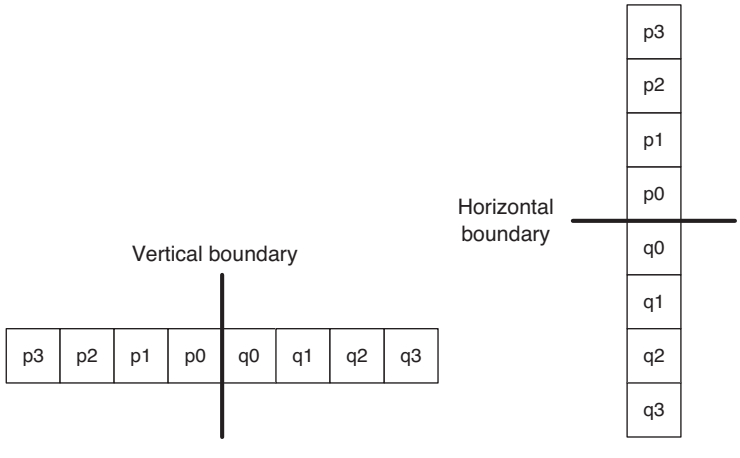


Figure 6.46 Pixels adjacent to vertical and horizontal boundaries

The filter is stronger at places where there is likely to be significant blocking distortion, such as the boundary of an intra coded macroblock or a boundary between blocks that contain coded coefficients.

6.5.2 Filter decision

A group of samples from the set (p2,p1,p0,q0,q1,q2) is filtered only if:

(a) Bs > 0 and
(b) |p0-q0|, |p1-p0| and |q1-q0| are each **less** than a threshold α or β, where α and β are defined in the standard.

The thresholds α and β increase with the average quantizer parameter QP of the two blocks p and q. The purpose of the filter decision is to 'switch off' the filter when there is a significant change or gradient across the block boundary in the original image. When QP is small, anything other than a very small gradient across the boundary is likely to be due to image features that should be preserved and not due to blocking effects and so the thresholds α and β are low. When QP is larger, blocking distortion is likely to be more significant and α, β are higher so that more boundary pixels are filtered.

Example

Figure 6.47 shows the 16 × 16 luma component of a macroblock without any blocking distortion with four 4 × 4 blocks a,b,c and d highlighted. Assuming a medium to large value of QP, the block boundary between a and b is likely to be filtered because the gradient across this boundary is small. There are no significant image features to preserve and blocking distortion will be quite obvious on this boundary. However, there is a significant change in luminance across the boundary between c and d due to a horizontal image feature and so the filter is switched off to preserve this strong filter.

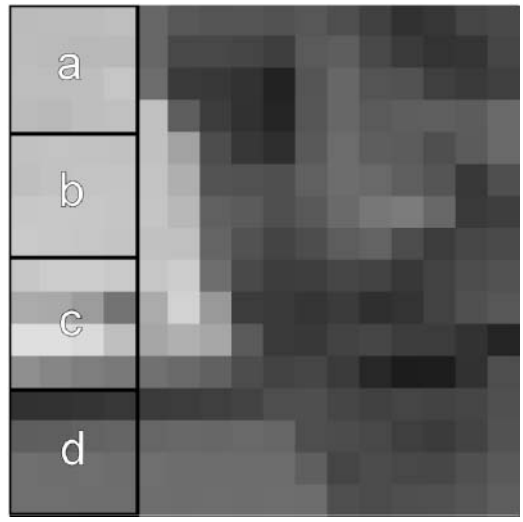


Figure 6.47 16 × 16 luma macroblock showing block edges

6.5.3 Filter implementation

(a) Bs ∈ {1,2,3}:
A 4-tap linear filter is applied with inputs p1, p0, q0 and q1, producing filtered outputs P0 and Q0 (0<Bs<4). If |p2-p0| is less than a threshold β, another 4-tap linear filter is applied with inputs p2, p1, p0 and q0, producing filtered output P1. If |q2-q0| is less than the threshold β, a 4-tap linear filter is applied with inputs q2, q1, q0 and p0, producing filtered output Q1. p1 and q1 are never filtered for chroma, only for luma data.

(b) Bs = 4:
If |p2-p0|<β and |p0-q0|<round(α/4):
P0 is produced by 5-tap filtering of p2, p1, p0, q0 and q1
P1 is produced by 4-tap filtering of p2, p1, p0 and q0
Luma only: P2 is produced by 5-tap filtering of p3, p2, p1, p0 and q0.
else:
P0 is produced by 3-tap filtering of p1, p0 and q1.

If |q2-q0|<β and |p0-q0|<round(α/4):
Q0 is produced by 5-tap filtering of q2, q1, q0, p0 and p1
Q1 is produced by 4-tap filtering of q2, q1, q0 and p0
Luma only: Q2 is produced by 5-tap filtering of q3, q2, q1, q0 and p0.
else:
Q0 is produced by 3-tap filtering of q1, q0 and p1.

6.5.4 Loop filter example

A video clip is encoded with a fixed Quantization Parameter of 36, i.e. relatively high quantization. Figure 6.48 shows an original frame from the clip and Figure 6.49 shows the same

Figure 6.48 Original frame, 'violin' frame 2

Figure 6.49 Reconstructed, QP = 36, no filter

frame after inter coding and decoding, with the loop filter disabled. Note the obvious blocking artefacts and note also the effect of varying motion-compensation block sizes for example, 16 × 16 blocks in the background to the left of the picture; 4 × 4 blocks around the arm. With the loop filter enabled (Figure 6.50) there is still some obvious distortion but most of the block edges have disappeared or faded. Note that sharp contrast boundaries such as the line of the arm against the dark piano are preserved by the filter whilst block edges in smoother regions of the picture such as the background to the left are smoothed. In this example the

Figure 6.50 Reconstructed, QP = 36, with filter

Figure 6.51 Reconstructed, QP = 32, no filter

loop filter makes only a small contribution to compression efficiency: the encoded bitrate is around 1.5 per cent smaller and the PSNR around 1 per cent larger for the sequence with the filter. However, the subjective quality of the filtered sequence is significantly better. The coding performance gain provided by the filter depends on the bitrate and sequence content.

Figure 6.51 and Figure 6.52 show the un-filtered and filtered frame respectively, this time with a lower quantizer parameter (QP = 32).

Figure 6.52 Reconstructed, QP = 32, with filter

6.6 Summary

The first stage of redundancy removal in an H.264/AVC encoder is the prediction process. Prediction involves creating an estimate of the current block of data from previously-coded image samples which is then subtracted from the current block to reduce its information content. An H.264/AVC decoder generates the same prediction and adds it to the decoded residual data.

Every macroblock is predicted, using intra prediction, in which the estimate is created from samples within the same frame or field, or inter prediction, in which the estimate is created from samples in a previously coded frame or field. An H.264/AVC encoder has many choices of different intra- or inter-prediction modes. Making the 'right' choice will result in an efficient prediction and a minimal residual.

After prediction, the residual data is transformed, quantized and coded prior to transmission or storage. The next chapter examines these stages.

6.7 References

i. M. Flierl and B. Girod, 'Generalized B pictures and the draft H.264/AVC video compression standard', *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, July 2003, pp. 587–597.

ii. T. Wedi and H. G. Mussman, 'Motion- and aliasing-compensated prediction for hybrid video coding', *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, July 2003, pp. 577–586.

iii. H. Schwarz, D. Marpe and T. Wiegand, 'Analysis of Hierarchical B Pictures and MCTF', IEEE International Conference on Multimedia and Expo (2006), pp. 1929–1932.

iv. P. List, A. Joch, J. Lainema, G. Bjontegaard and M. Karczewicz, 'Adaptive Deblocking Filter', *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, July 2003, pp. 614–619.

7

H.264 transform and coding

7.1 Introduction

Chapter 6 described the 'front end' of an H.264 encoder, the prediction processes that remove some redundancy by creating and subtracting an estimate of the current block of image data. This 'front end' prediction stage is lossless, i.e. it is a process that is fully reversible without loss of data. However, H.264 is fundamentally a lossy compression format, in which a degree of visual distortion is introduced into the video signal as a trade-off for higher compression performance. This distortion occurs in the transform/quantization process. In earlier standards there was an obvious boundary between the transform, converting a block of image samples into a different domain, and quantization, reducing the precision of transform coefficients. As will become clear, this boundary is less obvious in an H.264 codec, with an overlap of the transform and quantization stages. This, together with the new approach of exactly specifying a reversible integer transform 'core', makes the H.264 transform and quantization stage significantly different from earlier compression standards.

After prediction, transform and quantization, the video signal is represented as a series of quantized transform coefficients together with prediction parameters. These values must be coded into a bitstream that can be efficiently transmitted or stored and can be decoded to reconstruct the video signal. H.264/AVC provides several different mechanisms for converting parameters into a compressed bitstream, namely: fixed length binary codes, variable length Exponential-Golomb codes, context adaptive variable length codes (CAVLC) and context adaptive binary arithmetic coding (CABAC).

In this chapter we develop the H.264 transform and quantization processes and show how binary coding completes the process of converting a source video signal into an H.264/AVC compressed bitstream.

7.2 Transform and quantization

7.2.1 The H.264 transforms

H.264/AVC specifies transform and quantization processes that are designed to provide efficient coding of video data, to eliminate mismatch or 'drift' between encoders and decoders and to facilitate low complexity implementations.

The H.264 Advanced Video Compression Standard, Second Edition Iain E. Richardson
© 2010 John Wiley & Sons, Ltd

Earlier standards for image and video compression such as JPEG [i], MPEG-2 Video [ii] and MPEG-4 Visual [iii] specify a two-dimensional Discrete Cosine Transform (2-D DCT) to be applied to source or residual image data. In these standards, the transform is defined as an equation. For example, (7.1) defines a two-dimensional inverse DCT for blocks of size N × N, where Y_{xy} are input coefficients and X_{ij} are output image or residual samples.

$$X_{ij} = \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} C_x C_y Y_{xy} \cos \frac{(2j+1)y\pi}{2N} \cos \frac{(2i+1)x\pi}{2N} \tag{7.1}$$

Implementation of (7.1) for N>2 on a practical processor requires approximations to certain irrational multiplication factors, $\cos \frac{a\pi}{2N}$. Different approximations can significantly change the output of the forward or inverse transform, leading to mismatch between different implementations of encoders and decoders. To limit the magnitude of this mismatch, the earlier standards specify that the inverse transform must meet accuracy criteria based on IEEE Standard 1180-1990 [iv]. Nevertheless, there is still likely to be a mismatch between inverse DCTs in an encoder, which must carry out the inverse DCT to reconstruct frames for inter prediction, and a decoder. This leads to discrepancies between the prediction references in encoder and decoder and to 'drift' or cumulative distortion in the decoder output. In MPEG-2 Video and MPEG-4 Visual, this is mitigated by ensuring that coded blocks are periodically refreshed by intra coding.

In H.264/AVC and in other recent standards such as VC-1 [v], the transform and quantization processes are designed to minimize computational complexity, to be suitable for implementation using limited-precision integer arithmetic and to avoid encoder/decoder mismatch [vi, vii]. This is achieved by:

- using a core transform, an integer transform, that can be carried out using integer or fixed-point arithmetic and,
- integrating a normalization step with the quantization process to minimize the number of multiplications required to process a block of residual data.

The scaling and inverse transform processes carried out by a decoder are exactly specified in the standard so that every H.264 implementation should produce identical results, eliminating mismatch between different transform implementations.

7.2.2 Transform processes

The following sections describe the forward and inverse transform/quantization processes for luma and chroma coefficients.

7.2.2.1 Overview of transform processes

The inverse transform and re-scaling or 'inverse quantization' processes shown in Figure 7.1, are defined in the H.264/AVC standard. These processes or their equivalents must be

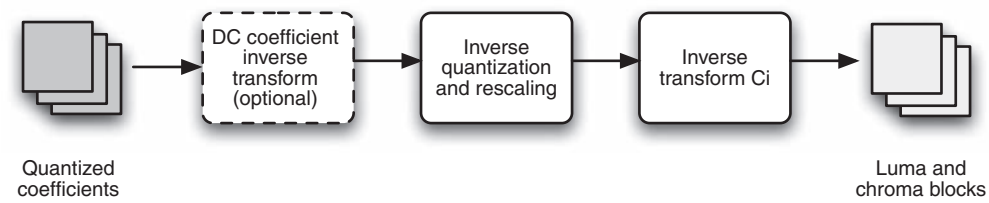


Figure 7.1 Re-scaling and inverse transform

implemented in every H.264-compliant decoder. The corresponding forward transform and quantization processes are not standardized but equivalent processes can be derived from the inverse transform / rescaling processes (Figure 7.2).

In an H.264 encoder, a block of residual coefficients is transformed and quantized (Figure 7.2). The basic transform, 'core transform', is a 4 × 4 or 8 × 8 **integer transform**, a scaled approximation to the Discrete Cosine Transform, DCT. In certain cases, part of the output of this integer transform is further transformed, 'DC transform', using a Hadamard Transform. The transform coefficients are scaled and quantized.

The corresponding inverse processes are shown in Figure 7.1. The DC inverse transform, if present, is carried out **before** rescaling. The rescaled coefficients are inverse transformed with a 4 × 4 or 8 × 8 inverse integer transform.

7.2.2.2 Luma transform processes

The default forward processes for luma samples shown in Figure 7.3 are applied **except** where (a) the macroblock is coded using 16 × 16 Intra Prediction (Chapter 6) or (b) an 8 × 8 integer transform is selected for the macroblock, in the High profiles only, see Chapter 8. In the default process, each 4 × 4 block within the 16 × 16 luma region of the macroblock is transformed (C_{f4}), scaled and quantized (M_{f4}) to produce a block of 4 × 4 quantized transform coefficients. The coefficient blocks are coded and transmitted in the order shown, from 0 to 15. The corresponding inverse processes are shown in Figure 7.4.

If the macroblock is predicted using 16 × 16 Intra Prediction (Figure 7.5), a second transform is applied to the lowest or 'DC' frequency coefficients of the first transform. These DC values tend to be highly correlated and this second transform improves the coding performance. First, C_{f4} is applied to each 4 × 4 block of samples. Next, the DC coefficients of every 4 × 4 block of coefficients are collected to form a 4 × 4 DC coefficient block. This DC coefficient block

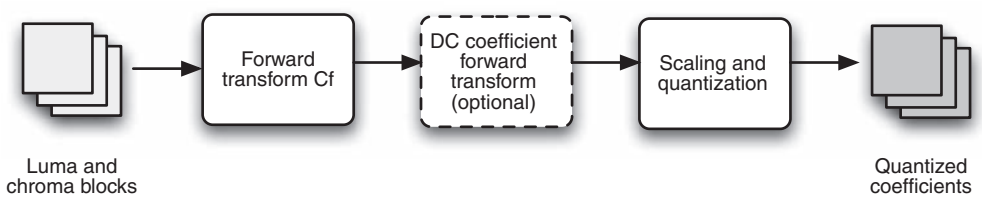


Figure 7.2 Forward transform and quantization

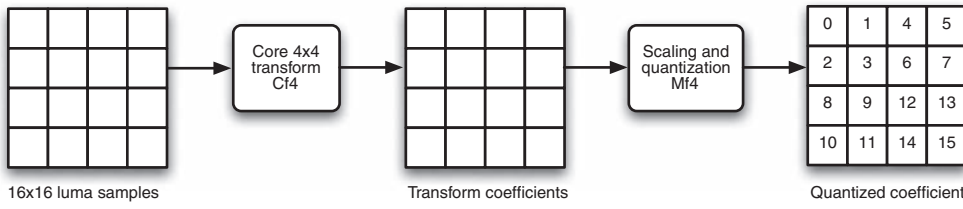


Figure 7.3 Luma forward transform : default

is further transformed using a 4 × 4 Hadamard transform. The transformed DC coefficient block and the remaining 15 AC coefficients in each block are scaled and quantized (M_{f4}) and transmitted in the order shown in Figure 7.5. The inverse processes carried out at the decoder are shown in Figure 7.6. Note that the inverse 4 × 4 DC transform occurs **before** scaling and inverse quantization – this is done in order to optimize the dynamic range during the inverse transform process [vi].

If the optional 8 × 8 transform is enabled for this macroblock, only available in High profiles, for macroblocks that are coded using Intra 8 × 8 prediction or any Inter prediction mode, the processes shown in Figure 7.7 are applied. Each 8 × 8 block of luma samples is transformed (C_{f8}), scaled and quantized (M_{f8}) and transmitted as shown, with the corresponding inverse processes in Figure 7.8.

7.2.2.3 Chroma transform processes

The chroma components of a macroblock in 4:2:0 format, 16 × 16 luma samples with 8 × Cr and 8 × 8 Cb samples, are processed as shown in Figure 7.9. Each 4 × 4 block of Cb or Cr samples is transformed (C_{f4}). The four DC coefficients of each block are further transformed with a 2 × 2 Hadamard or DCT transform. The two DC blocks, labelled 0 and 1, followed by the AC blocks 2 to 9 are scaled, quantized and transmitted. Note that the inverse DC transform is applied before scaling and inverse quantizing during the inverse processes (Figure 7.10).

A macroblock in 4:2:2 format contains 8 × 16 Cb and Cr samples (Figure 7.11). Each 4 × block is transformed (C_{f4}) and the eight DC coefficients are further transformed with a 2 × 4 Hadamard transform. The two DC blocks 0 and 1 followed by the AC blocks 2 to 17 are scaled, quantized and transmitted. The inverse processes are shown in Figure 7.12 – again, note the order of the DC inverse transform and scaling/inverse quantization.

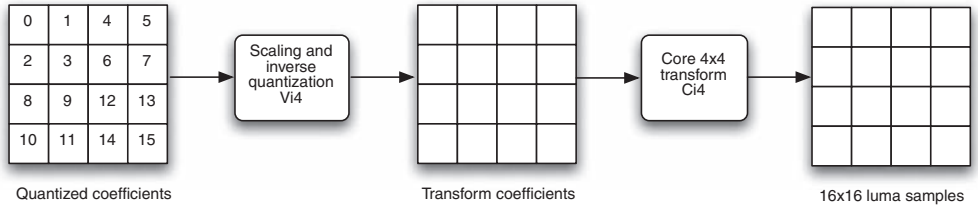


Figure 7.4 Luma inverse transform : default

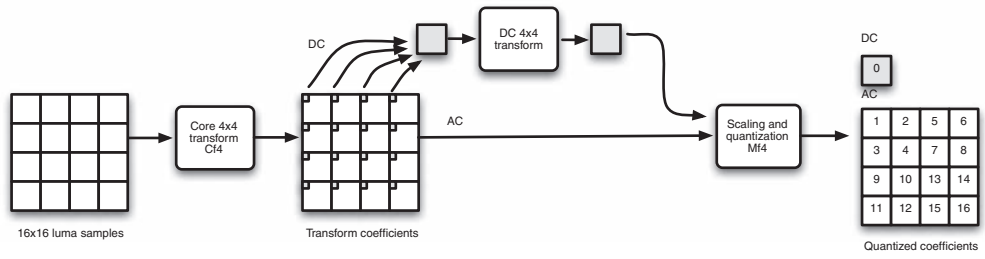


Figure 7.5 Luma forward transform : Intra 16 × 16 mode

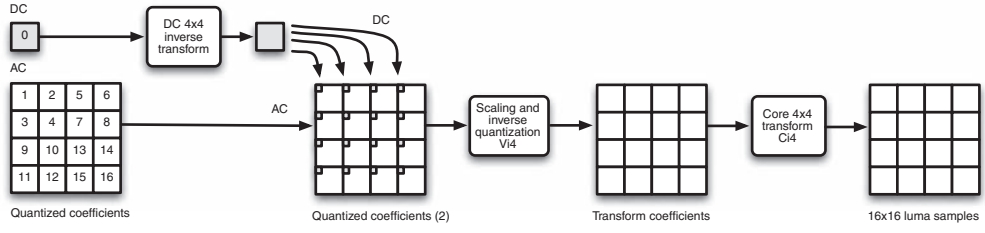


Figure 7.6 Luma inverse transform : Intra 16 × 16 mode

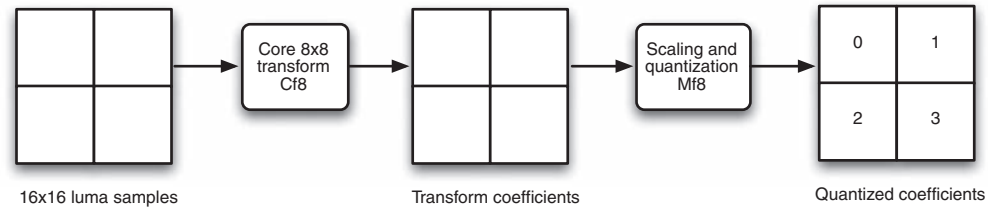


Figure 7.7 Luma forward transform : 8 × 8 transform

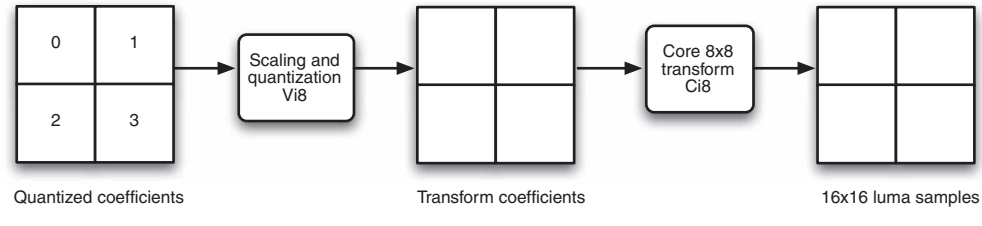


Figure 7.8 Luma inverse transform : 8 × 8 transform

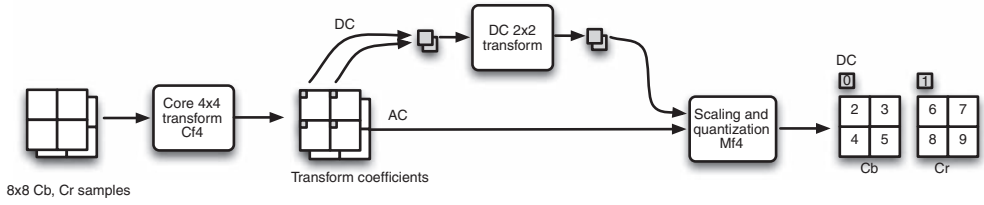


Figure 7.9 Chroma forward transform : 4:2:0 macroblock

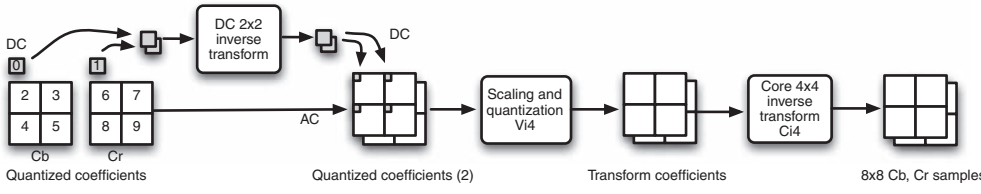


Figure 7.10 Chroma inverse transform : 4:2:0 macroblock

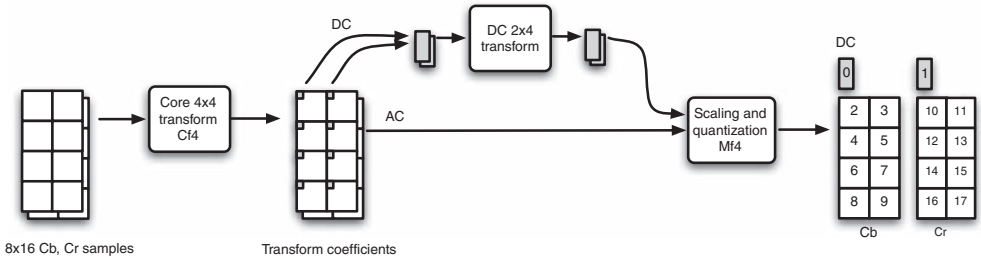


Figure 7.11 Chroma forward transform : 4:2:2 macroblock

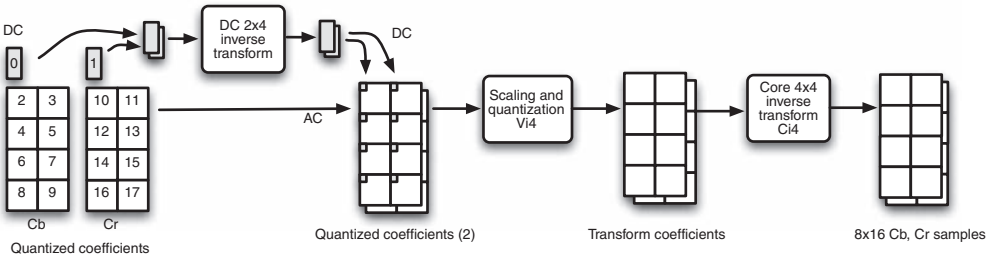


Figure 7.12 Chroma inverse transform : 4:2:2 macroblock

The chroma components in a 4:4:4-format macroblock are the same size as the luma component, with 16 × 16 samples each in Cb and Cr. Each chroma component is processed as shown in Figure 7.3 (default), Figure 7.5 if the macroblock is coded using 16 × 16 Intra prediction or Figure 7.7 if the 8 × 8 transform is selected and/or the macroblock is coded using 8 × 8 Intra prediction.

7.2.3 Integer transform and quantization : 4 × 4 blocks

The forward and inverse integer transform processes for 4 × 4 blocks are developed as follows.

- Starting from a 4 × 4 DCT, derive a scaled, rounded integer approximation to the DCT.
- Add a normalization step to maintain the orthonormal property of the DCT.
- Integrate the normalization step with a quantization process.
- Specify the inverse scaling and quantization process, defined in the standard. Derive an equivalent forward scaling and quantization process.

7.2.3.1 Developing the forward transform and quantization process

(a) Consider a block of pixel data that is processed by a two-dimensional Discrete Cosine Transform (DCT) followed by quantization, i.e. rounded division by a quantization step size, Q_{step} (Figure 7.13a).

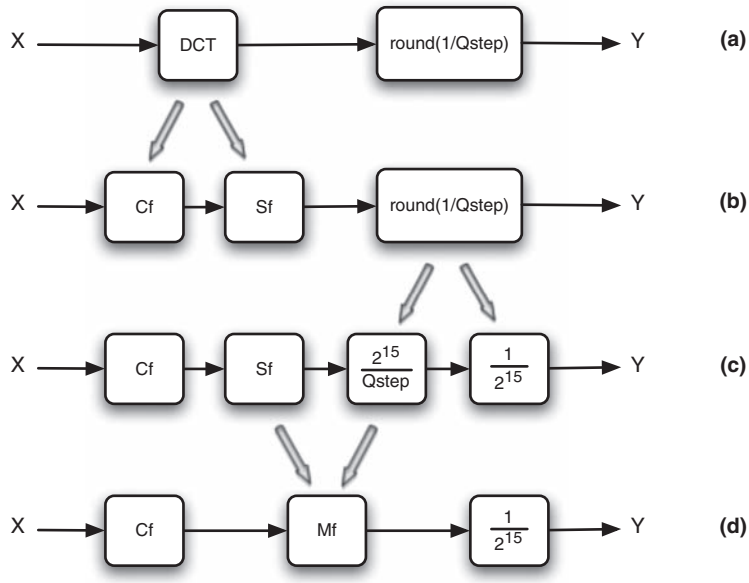


Figure 7.13 Development of the forward transform and quantization process

(b) Rearrange the DCT process into a core transform ($\mathbf{C_{f4}}$) and a scaling matrix ($\mathbf{S_{f4}}$) (Figure 7.13b).
(c) Scale the quantization process by a constant (2^{15}) and compensate by dividing and rounding the final result (Figure 7.13c). The constant factor 2^{15} is chosen as a compromise between higher accuracy and limited arithmetic precision.
(d) Combine $\mathbf{S_{f4}}$ and the quantization process into $\mathbf{M_{f4}}$ (Figure 7.13d), where:

$$\mathbf{M}_f \approx \frac{\mathbf{S}_f \cdot 2^{15}}{Q_{step}} \tag{7.2}$$

Note that $\mathbf{M}_\mathrm{f}$ is actually derived from parameter $\boldsymbol{V}_\mathrm{i}$ (see next section), as described in section 7.2.3.7.

7.2.3.2 Developing the rescaling and inverse transform process

(a) Consider a re-scaling or 'inverse quantization' operation followed by a two-dimensional inverse DCT (IDCT) (Figure 7.14a).
(b) Rearrange the IDCT process into a core transform ($\mathbf{C_i}$) and a scaling matrix ($\mathbf{S_i}$) (Figure 7.14b).
(c) Scale the re-scaling process by a constant (2^6) and compensate by dividing and rounding the final result (Figure 7.14c). Note that rounding need not be to the nearest integer.
(d) Combine the re-scaling process and $\mathbf{S_i}$ into $\mathbf{V_i}$ (Figure 7.14d), where:

$$V_i \approx S_i \cdot 2^6 \cdot Q_{step} \tag{7.3}$$

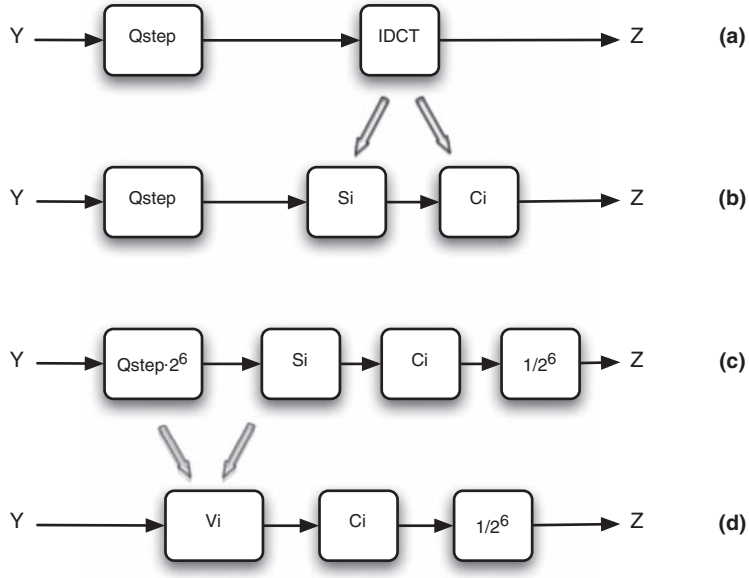


Figure 7.14 Development of the rescaling and inverse transform process

7.2.3.3 Developing $\mathbf{C_{f4}}$ and $\mathbf{S_{f4}}$: 4 × 4 blocks

Consider a 4 × 4 two-dimensional DCT of a block **X**:

$$\mathbf{Y} = \mathbf{A} \cdot \mathbf{X} \cdot \mathbf{A}^{\mathbf{T}} \tag{7.4}$$

Where · indicates matrix multiplication and:

$$\mathbf{A} = \begin{bmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & -c \end{bmatrix}, \quad \begin{array}{l} a = {}^{1}/_{2} \\ b = \sqrt{{}^{1}/_{2}} \cos {}^{\pi}/_{8} = 0.6532\ldots \\ c = \sqrt{{}^{1}/_{2}} \cos {}^{3\pi}/_{8} = 0.2706\ldots \end{array}$$

Note that the rows of **A** are orthogonal and have unit norms, i.e. the rows are orthonormal, a necessary condition for an orthogonal block transform.

Calculation of [set as equation] (7.4 on a practical processor requires approximation of the irrational numbers b and c. A fixed-point approximation is equivalent to scaling each row of **A** and rounding to the nearest integer. Choosing a particular approximation, namely multiply by ∼2.5 and round, gives $\mathbf{C_{f4}}$:

$$\mathbf{C}_{f4} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \tag{7.5}$$

This approximation is chosen to minimize the complexity of implementing the transform, since multiplication by $\mathbf{C_{f4}}$ requires only additions and binary shifts, whilst maintaining good compression performance [vi].

The row norms of $\mathbf{C_{f4}}$ are $\neq 1$. To restore the orthonormal property of the original matrix **A**, multiply all the values c_{ij} in row r by $\frac{1}{\sqrt{\sum_j c_{rj}^2}}$:

$$\mathbf{A}_1 = \mathbf{C_{f4}} \bullet \mathbf{R_{f4}} \quad \text{where } \mathbf{R}_{f4} = \begin{bmatrix} 1/2 & 1/2 & 1/2 & 1/2 \\ 1/\sqrt{10} & 1/\sqrt{10} & 1/\sqrt{10} & 1/\sqrt{10} \\ 1/2 & 1/2 & 1/2 & 1/2 \\ 1/\sqrt{10} & 1/\sqrt{10} & 1/\sqrt{10} & 1/\sqrt{10} \end{bmatrix} \tag{7.6}$$

• denotes element-by-element multiplication, the Hadamard or Schur product, where $\mathbf{P} = \mathbf{Q} \bullet \mathbf{R}$ means that each element $\mathrm{p_{ij}} = \mathrm{q_{ij}} \cdot \mathrm{r_{ij}}$. Note that the new matrix $\mathbf{A_1}$ is orthonormal.

The two-dimensional transform (7.4) becomes:

$$\mathrm{Y} = \mathrm{A}_1 \cdot \mathrm{X} \cdot \mathrm{A}_1^{\mathrm{T}} = [\mathrm{C_{f4}} \bullet \mathrm{R_{f4}}] \cdot \mathrm{X} \cdot [\mathrm{C}_{\mathrm{f4}}^{\mathrm{T}} \bullet \mathrm{R}_{\mathrm{f4}}^{\mathrm{T}}] \tag{7.7}$$

Rearranging to extract the scaling arrays $\mathrm{R_{f4}}$:

$$\begin{aligned} \mathbf{Y} &= [\mathbf{C_{f4}} \cdot \mathbf{X} \cdot \mathbf{C}_{\mathbf{f4}}^{\mathbf{T}}] \bullet [\mathbf{R_{f4}} \bullet \mathbf{R}_{\mathbf{f4}}^{\mathbf{T}}] \\ &= [\mathbf{C_{f4}} \cdot \mathbf{X} \cdot \mathbf{C}_{\mathbf{f4}}^{\mathbf{T}}] \bullet \mathbf{S_{f4}} \end{aligned} \tag{7.8}$$

Where

$$\mathbf{S}_{f4} = \mathbf{R}_{f4} \bullet \mathbf{R}_{f4}^{T} = \begin{bmatrix} 1/4 & 1/2\sqrt{10} & 1/4 & 1/2\sqrt{10} \\ 1/2\sqrt{10} & 1/10 & 1/2\sqrt{10} & 1/10 \\ 1/4 & 1/2\sqrt{10} & 1/4 & 1/2\sqrt{10} \\ 1/2\sqrt{10} & 1/10 & 1/2\sqrt{10} & 1/10 \end{bmatrix}$$

7.2.3.4 Developing $\mathbf{C_{i4}}$ and $\mathbf{S_{i4}}$: 4 × 4 blocks

Consider a 4 × 4 two-dimensional IDCT of a block **Y**:

$$\mathbf{Z} = \mathbf{A^T} \cdot \mathbf{Y} \cdot \mathbf{A} \tag{7.9}$$

Where

$$\mathbf{A} = \begin{bmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & -c \end{bmatrix}, \quad \begin{array}{l} a = {}^{1}/_{2} \\ b = \sqrt{{}^{1}/_{2}} \cos {}^{\pi}/_{8} = 0.6532\ldots \\ c = \sqrt{{}^{1}/_{2}} \cos {}^{3\pi}/_{8} = 0.2706\ldots \end{array} \quad \text{as before.}$$

Choose a particular approximation by scaling each row of **A** and rounding to the nearest 0.5, giving $\mathbf{C_i}$:

$$\mathbf{C}_{i4} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1/2 & -1/2 & -1 \\ 1 & -1 & -1 & 1 \\ 1/2 & -1 & 1 & -1/2 \end{bmatrix} \tag{7.10}$$

Again, this approximation is chosen as a trade-off between computational simplicity and compression performance. The dynamic range of the inputs to C_{i4} is greater than that of the corresponding inputs to C_{f4} and so a smaller scaling factor is used for the second and fourth rows of C_{i4}. Compare with C_{f4} in section 7.2.3.3 [vi]. To restore orthonormality, multiply all the values c_{ij} in row r by $\frac{1}{\sqrt{\sum_j c_{rj}^2}}$:

$$\mathbf{A}_2 = \mathbf{C_{i4}} \bullet \mathbf{R_{i4}} \quad \text{where } \mathbf{R}_{i4} = \begin{bmatrix} 1/2 & 1/2 & 1/2 & 1/2 \\ \sqrt{2/5} & \sqrt{2/5} & \sqrt{2/5} & \sqrt{2/5} \\ 1/2 & 1/2 & 1/2 & 1/2 \\ \sqrt{2/5} & \sqrt{2/5} & \sqrt{2/5} & \sqrt{2/5} \end{bmatrix} \tag{7.11}$$

The two-dimensional inverse transform (7.9) becomes:

$$\mathbf{Z} = \mathbf{A_2^T} \cdot \mathbf{Y} \cdot \mathbf{A}_2 = [\mathbf{C_{i4}^T} \bullet \mathbf{R_{i4}^T}] \cdot \mathbf{Y} \cdot [\mathbf{C_{i4}} \bullet \mathbf{R_{i4}}] \tag{7.12}$$

Rearranging:

$$\begin{aligned}\mathbf{Z} &= [\mathbf{C}_{\mathbf{i}4}^{\mathbf{T}}] \cdot [\mathbf{Y} \bullet \mathbf{R}_{\mathbf{i}4}^{\mathbf{T}} \bullet \mathbf{R}_{\mathbf{i}}4] \cdot [\mathbf{C}_{\mathbf{i}}4] \\ &= [\mathbf{C}_{\mathbf{i}4}^{\mathbf{T}}] \cdot [\mathbf{Y} \bullet \mathbf{S}_{\mathbf{i}}4] \cdot [\mathbf{C}_{\mathbf{i}}4]\end{aligned} \tag{7.13}$$

Where

$$\mathbf{S}_{i4} = \mathbf{R}_{i4}^{T} \bullet \mathbf{R}_{i4} = \begin{bmatrix} 1/4 & 1/\sqrt{10} & 1/4 & 1/\sqrt{10} \\ 1/\sqrt{10} & 2/5 & 1/\sqrt{10} & 2/5 \\ 1/4 & 1/\sqrt{10} & 1/4 & 1/\sqrt{10} \\ 1/\sqrt{10} & 2/5 & 1/\sqrt{10} & 2/5 \end{bmatrix}$$

The core inverse transform $\mathbf{C_{i4}}$ and the rescaling matrix $\mathbf{V_{i4}}$ are defined in the H.264 standard. We now develop $\mathbf{V}_{i4}$ and will then derive the forward scaling matrix $\mathbf{M_{f4}}$ from $\mathbf{V_{i4}}$.

7.2.3.5 Developing $\mathbf{V_{i4}}$

From (7.3),

$$\mathbf{V}_{\mathbf{i}4} \approx \mathbf{S}_{\mathbf{i}4} \cdot \mathrm{Q}_{\mathrm{step}} \cdot 2^6 \tag{7.14}$$

H.264 supports a range of 'effective' quantization step sizes $\mathrm{Q}_{\mathrm{step}}$. The actual step sizes are not defined in the standard, rather the scaling matrix $\mathbf{V_{i4}}$ is specified as a function of QP. An optional scaling matrix may be used to weight the quantization operation depending on the transform coefficient position, see section 7.2.6.

The values in the matrix $\mathbf{V_{i4}}$ depend on $\mathrm{Q}_{\mathrm{step}}$ and hence QP and on the scaling factor matrix $\mathbf{S_{i\,4}}$. These are shown for QP 0 to 5 in Table 7.1.

For higher values of QP, the corresponding values in $\mathbf{V_{i4}}$ are doubled, i.e. $\mathbf{V_{i4}}$ (QP = 6) = $2\mathbf{V_{i4}}$(QP = 0), etc.

Note from Table 7.1 that there are only three unique values in each matrix $\mathbf{V_{i4}}$. These three values are defined as a table of values $\boldsymbol{v}$ in the H.264 standard, for QP = 0 to QP = 5 (Table 7.2).

Hence for QP values from 0 to 5, $\mathbf{V_{i4}}$ is obtained as:

$$V_{i4} = \begin{bmatrix} v(QP,0) & v(QP,2) & v(QP,0) & v(QP,2) \\ v(QP,2) & v(QP,1) & v(QP,2) & v(QP,1) \\ v(QP,0) & v(QP,2) & v(QP,0) & v(QP,2) \\ v(QP,2) & v(QP,1) & v(QP,2) & v(QP,1) \end{bmatrix} \tag{7.15}$$

Denote this as:

$$\mathbf{V}_{\mathbf{i}4} = v(QP, n)$$

Where v (r,n) is row r, column n of v.

Table 7.1 $V_{i\,4}$ values, 4×4 blocks

QP	$\mathbf{V_{i4}} \approx \text{round}\,(\mathbf{S_{i4}} \cdot Q_{step} \cdot 2^6)$
0	$\begin{bmatrix} 10 & 13 & 10 & 13 \\ 13 & 16 & 13 & 16 \\ 10 & 13 & 10 & 13 \\ 13 & 16 & 13 & 16 \end{bmatrix}$
1	$\begin{bmatrix} 11 & 14 & 11 & 14 \\ 14 & 18 & 14 & 18 \\ 11 & 14 & 11 & 14 \\ 14 & 18 & 14 & 18 \end{bmatrix}$
2	$\begin{bmatrix} 13 & 16 & 13 & 16 \\ 16 & 20 & 16 & 20 \\ 13 & 16 & 13 & 16 \\ 16 & 20 & 16 & 20 \end{bmatrix}$
3	$\begin{bmatrix} 14 & 18 & 14 & 18 \\ 18 & 23 & 18 & 23 \\ 14 & 18 & 14 & 18 \\ 18 & 23 & 18 & 23 \end{bmatrix}$
4	$\begin{bmatrix} 16 & 20 & 16 & 20 \\ 20 & 25 & 20 & 25 \\ 16 & 20 & 16 & 20 \\ 20 & 25 & 20 & 25 \end{bmatrix}$
5	$\begin{bmatrix} 18 & 23 & 18 & 23 \\ 23 & 29 & 23 & 29 \\ 18 & 23 & 18 & 23 \\ 23 & 29 & 23 & 29 \end{bmatrix}$

Table 7.2 Table *v* defined in H.264 standard

QP	***v*** (r, 0): $\mathbf{V_{i4}}$ positions (0,0), (0,2), (2,0), (2,2)	***v*** (r, 1): $\mathbf{V_{i4}}$ positions (1,1), (1,3), (3,1), (3,3)	***v*** (r, 2): Remaining $\mathbf{V_{i4}}$ positions
0	10	16	13
1	11	18	14
2	13	20	16
3	14	23	18
4	16	25	20
5	18	29	23

Table 7.3 Estimated Q_{step} (4 × 4 blocks) = V_{i4} / (Si · 2^6), element-by-element division

QP	~Q_{step} (n = 0)	~Q_{step} (n = 1)	~Q_{step} (n = 2)
0	0.625	0.625	0.6423
1	0.6875	0.7031	0.6917
2	0.8125	0.7812	0.7906
3	0.875	0.8984	0.8894
4	1.0	0.9766	0.9882
5	1.125	1.1328	1.1364
6	1.25	1.25	1.2847
...	...		
12	2.5	2.5	2.5694
...	...		
18	5.0	5.0	5.1387
...	...		
48	160	160	164.4384
...	...		
51	224	230	227.6840

For values of QP>5, index the row of array v by QP%6 and then multiply by $2^{\text{floor(QP/6)}}$. In general:

$$\mathbf{V_{i4}} = v(\text{QP\%6, n}) \cdot 2^{\text{floor(QP/6)}} \tag{7.16}$$

Working back from $\mathbf{V_{i4}}$, the equivalent quantizer step size Q_{step} can be estimated (Table 7.3). Note that different estimated values, ~Q_{step} are obtained for the three distinct columns n of array v.

The ratio between successive Q_{step} values in Table 7.3 is approximately $\sqrt[6]{2} = 1.2246\cdots$ so that Q_{step} doubles in size when QP increases by 6. Any value of Q_{step} can be derived from the first 6 values in the table, QP0 − QP5, as follows:

$$Q_{step}(\text{QP}) = Q_{step}(\text{QP\%6}) \cdot 2^{\text{floor(QP/6)}} \tag{7.17}$$

Figure 7.15 plots the relationship between QP (x-axis) and effective Q_{step} for array positions corresponding to column 0 of v (y-axis). Note the logarithmic scale on the y-axis. The circles are plotted at QPs of 0, 6, 12, . . ., 48.

7.2.3.6 The complete 4 × 4 inverse transform and scaling process

The complete inverse transform and scaling process for 4 × 4 blocks in macroblocks excluding 16 × 16-Intra mode, is:

$$\mathbf{Z} = \text{round}\left([\mathbf{C_{i4}^T}] \cdot [\mathbf{Y} \bullet \boldsymbol{v}(\text{QP\%6, n}) \cdot 2^{\text{floor(QP/6)}}] \cdot [\mathbf{C_{i4}}] \cdot \frac{1}{2^6}\right) \tag{7.18}$$

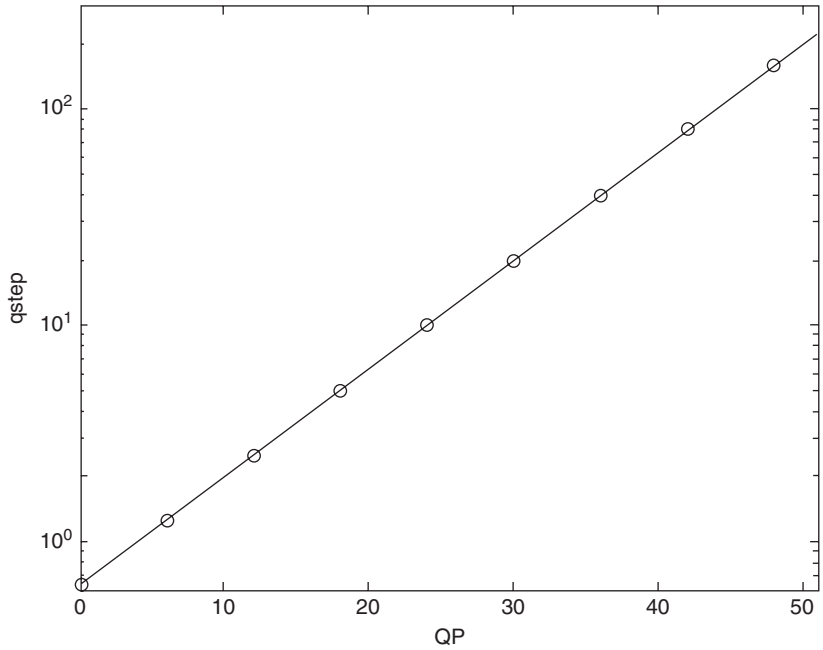


Figure 7.15 Quantization parameter QP vs. effective quantizer step size, logarithmic y-axis

In the H.264 standard, this process is described as follows:

1. Calculate a matrix LevelScale:

$$\text{LevelScale}(\text{QP}\%6, \text{i}, \text{j}) = \text{weightScale}(\text{i}, \text{j}) * \text{v}(\text{QP}\%6, \text{n})$$

   Where weightScale is a quantization scaling matrix, weightScale(i,j) $= 2^4 = 16$ by default[1].
2. Scale the input samples c_{ij}:
   if QP $> = 24$:

$$d_{ij} = (c_{ij} * \text{LevelScale}(\text{QP}\%6, \text{i}, \text{j})) \ll (\text{QP}/6 - 4)$$

   if QP < 24:

$$d_{ij} = (c_{ij} * \text{LevelScale}(\text{QP}\%6, \text{i}, \text{j}) + 2^{3-\text{QP}/6}) \gg (4 - \text{QP}/6)$$

[1] The default value of weightScale (16) is cancelled out by step 2 ($\gg 4$ or $\ll -4$) and is therefore redundant. Other weightScale tables may be used to weight the quantization parameter by different amounts depending on the matrix position (i,j). See section 7.2.6.

In this step, $\ll n$ means 'left shift by n bits' and division is rounded down. Steps 1 and 2 are equivalent to calculating:

$$\mathbf{D} = \mathbf{Y} \bullet \boldsymbol{v}(\mathrm{QP}\%6, \mathrm{n}) \cdot 2^{\mathrm{floor}(\mathrm{QP}/6)}$$

The pre-scaling by $2^4 = 16$ in step 1 is cancelled out in step 2 by effectively right-shifting each sample by 4 bit positions.

3. Compute the core transform:

$$\mathbf{H} = [\mathbf{C}_{\mathbf{i4}}^{\mathbf{T}}] \cdot \mathbf{D} \cdot [\mathbf{C}_{\mathbf{i4}}]$$

In the standard, the core transform is described as a series of arithmetic operations, additions, subtractions and bit shifts.

4. Divide each sample $\mathrm{h_{ij}}$ by 2^6:

$$\mathrm{r_{ij}} = (\mathrm{h_{ij}} + 2^5) \gg 6$$

This is equivalent to division by 2^6 and rounding towards zero.

The final array of residual samples $\mathrm{r_{ij}}$ is **R**, equivalent to **Z** in (7.18).

7.2.3.7 Deriving $\mathbf{M_{f4}}$

Combining (7.2) and (7.3):

$$\mathbf{M}_{f4} \approx \frac{\mathbf{S}_{i4} \bullet \mathbf{S}_{f4} \cdot 2^{21}}{\mathbf{V}_{i4}} \tag{7.19}$$

$\mathbf{S_{i4}}$, $\mathbf{S_{f4}}$ are known and $\mathbf{V}_{\mathrm{i4}}$ is defined in the standard, see section 7.2.3.5. Define $\mathbf{M}_{\mathrm{f4}}$ exactly as:

$$\mathbf{M}_{f4} = \mathrm{round}\left(\frac{\mathbf{S}_{i4} \bullet \mathbf{S}_{f4} \cdot 2^{21}}{\mathbf{V}_{i4}}\right) \tag{7.20}$$

The numerator of $\mathbf{M}_{\mathrm{f4}}$ is:

$$\mathbf{S}_{i4} \bullet \mathbf{S}_{f4} \cdot 2^{21} = \begin{bmatrix} 131072 & 104857.6 & 131072 & 104857.6 \\ 104857.6 & 83886.1 & 104857.6 & 83886.1 \\ 131072 & 104857.6 & 131072 & 104857.6 \\ 104857.6 & 83886.1 & 104857.6 & 83886.1 \end{bmatrix} \tag{7.21}$$

The entries in matrix $\mathbf{M_{f4}}$ may be listed as follows (Table 7.4):

Table 7.4 Tables v and m

QP	$\boldsymbol{v}$ (r, 0): $\mathbf{V_{i4}}$ positions (0,0), (0,2), (2,0), (2,2)	$\boldsymbol{v}$ (r, 1): $\mathbf{V_{i4}}$ positions (1,1), (1,3), (3,1), (3,3)	$\boldsymbol{v}$ (r, 2): Remaining $\mathbf{V_{i4}}$ positions	$\boldsymbol{m}$ (r, 0): $\mathbf{M}_{f4}$ positions (0,0), (0,2), (2,0), (2,2)	$\boldsymbol{m}$ (r, 1): $\mathbf{M}_{f4}$ positions (1,1), (1,3), (3,1), (3,3)	$\boldsymbol{m}$ (r,2): Remaining $\mathbf{M}_{f4}$ positions
0	10	16	13	13107	5243	8066
1	11	18	14	11916	4660	7490
2	13	20	16	10082	4194	6554
3	14	23	18	9362	3647	5825
4	16	25	20	8192	3355	5243
5	18	29	23	7282	2893	4559

Hence for QP values from 0 to 5, $\mathbf{M_{f4}}$ can be obtained from m, the last three columns of Table 7.4:

$$M_{f4} = \begin{bmatrix} m(QP,0) & m(QP,2) & m(QP,0) & m(QP,2) \\ m(QP,2) & m(QP,1) & m(QP,2) & m(QP,1) \\ m(QP,0) & m(QP,2) & m(QP,0) & m(QP,2) \\ m(QP,2) & m(QP,1) & m(QP,2) & m(QP,1) \end{bmatrix} \tag{7.22}$$

Denote this as:

$$\mathbf{M}_{\mathbf{f}4} = m(\mathrm{QP}, \mathrm{n})$$

Where m (r,n) is row r, column n of m.

For QP>5, index the row of array m by QP%6 and then divide by $2^{\mathrm{floor(QP/6)}}$. In general:

$$\mathbf{M_{f4}} = \boldsymbol{m}(\mathrm{QP\%6}, \mathrm{n})/2^{\mathrm{floor(QP/6)}} \tag{7.23}$$

Where $\boldsymbol{m}$ (r,n) is row r, column n of $\boldsymbol{m}$.

7.2.3.8 The complete 4 × 4 forward transform and scaling process

The complete forward transform, scaling and quantization process for 4 × 4 blocks and for modes excluding 16 × 16-Intra, becomes:

$$\begin{aligned} \mathbf{Y} &= \mathrm{round}\left([\mathbf{C_{f4}}] \cdot [\mathbf{X}] \cdot [\mathbf{C_{f4}^T}] \bullet \boldsymbol{m}(\mathrm{QP\%6}, \mathrm{n})/2^{\mathrm{floor(QP/6)}}] \cdot \frac{1}{2^{15}} \right) \\ &= \mathrm{round}\left([\mathbf{C_{f4}}] \cdot [\mathbf{X}] \cdot [\mathbf{C_{f4}^T}] \bullet \boldsymbol{m}(\mathrm{QP\%6}, \mathrm{n})] \cdot \frac{1}{2^{15+floor(QP/6)}} \right) \end{aligned} \tag{7.24}$$

Rounded division by 2^p may be carried out by adding an offset and right-shifting by p bit positions.

7.2.3.9 4 × 4 Transform and quantization: Examples

The forward and inverse transform and quantization processes are illustrated in the following examples. Note that all of the operations are calculated using integer arithmetic.

Core transform
Consider a block of luma or chroma samples, X:

58	64	51	58
52	64	56	66
62	63	61	64
59	51	63	69

Calculate $C_{f4} \cdot X$, a one-dimensional transform in the vertical direction:

231	242	231	257
−12	27	−29	−20
3	−12	−3	−3
19	11	−2	−15

Calculate $C_{f4} \cdot X \cdot C_{f4}{}^{T}$, the complete two-dimensional 'core' transform:

961	−41	15	−48
−34	72	−30	−104
−15	3	15	24
13	81	−5	8

Scaling and quantization, QP = 6
m (QP%6 = 0):

13107	8066	13107	8066
8066	5243	8066	5243
13107	8066	13107	8066
8066	5243	8066	5243

$[C_{f4}] \cdot [X] \cdot [C_{f4}{}^{T}] \bullet m$:

12595827	−330706	196605	−387168
−274244	377496	−241980	−545272
−196605	24198	196605	193584
104858	424683	−40330	41944

Quantized and scaled output Y, QP = 6, equivalent to qstep ≈ 1.25). The Y arrays are calculated using the rounding formula specified in the H.264 JM reference software model.

$$\mathbf{Y} = \text{round}\left([\mathbf{C_{f4}}] \cdot [\mathbf{X}] \cdot [\mathbf{C_{f4}^T}] \bullet \boldsymbol{m}(\text{QP\%6, n})] \cdot \frac{1}{2^{15+floor(QP/6)}} \right)$$

192	−5	3	−6
−4	5	−3	−8
−3	0	3	3
1	6	0	0

Decoded output Z using H.264 rescaling and inverse transform:

$$\mathbf{Z} = \text{round}\left([\mathbf{C_{i4}^T}] \cdot [\mathbf{Y} \bullet \boldsymbol{v}(\text{QP\%6, n}) \cdot 2^{\text{floor(QP/6)}}] \cdot [\mathbf{C_{i4}}] \cdot \frac{1}{2^6} \right)$$

58	63	51	59
53	64	57	66
62	63	60	64
59	52	63	68

Scaling and quantization, QP = 12

Quantized and scaled output Y, QP = 12, qstep ≈ 2.5. Note that the coefficients are approximately half the size of the QP = 6 outputs.

96	−2	1	−3
−2	3	−2	−4
−1	0	1	1
0	3	0	0

Decoded output Z:

57	65	51	57
53	64	57	65
62	62	59	63
59	53	64	69

Scaling and quantization, QP = 18
Quantized and scaled output Y, QP = 18, qstep ≈ 5. Coefficients are half the magnitude of the QP = 12 outputs.

48	−1	0	−1
−1	1	−1	−2
0	0	0	0
0	1	0	0

Decoded output Z:

55	66	54	58
54	62	58	63
61	59	61	62
60	55	65	67

Scaling and quantization, QP = 30
Quantized and scaled output Y, QP = 30, qstep ≈ 20.

12	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

Decoded output Z:

60	60	60	60
60	60	60	60
60	60	60	60
60	60	60	60

Note that all the AC coefficients are quantized to zero and so all the decoded block samples are identical.

Comparison with 4 × 4 DCT using floating-point arithmetic
A floating-point DCT such as Matlab's *dct2* uses approximations to the DCT coefficients which should be more accurate than the integer approximations in the H.264 core transform. Here, we compare the output of the H.264 transform and quantization process with the equivalent floating point DCT and quantization process.

Calculate the 4 × 4 DCT of X, using Matlab's floating point DCT function *dct2*:

240.250	−7.004	3.750	−7.111
−5.217	7.005	−4.788	−10.841
−3.750	0.742	3.750	3.752
2.431	7.659	−0.452	0.995

For comparison with the QP = 18 result, calculate the quantized coefficients using a quantizer step size of 5:

$$Y = \text{round}(\text{dct2}(X)/5) =$$

48	−1	1	−1
−1	1	−1	−2
−1	0	1	1
0	2	0	0

Decoded output Z_1, using rescaling or multiplication by 5 followed by Matlab's floating point *idct2*:

$$Z_1 = \text{round}(\text{idct2}(Y \times 5)) =$$

58	61	51	57
53	64	58	67
64	63	59	61
61	50	63	68

It is clear that the H.264 transform and quantization process, which uses a scaled integer DCT approximation, gives a similar but not identical result to a floating-point DCT approximation followed by quantization. The authors of [vi] show that in the context of a complete video codec, the performance of the H.264 transform and quantization is almost identical to that of higher-precision DCT implementations.

7.2.4 Integer transform and quantization : 8 × 8 blocks

The forward and inverse 8 × 8 transforms [viii] are developed in a similar way to the 4 × 4 integer transforms (section 7.2.3.1) but with the following differences:

1. The core transform $\mathbf{C_{f8}}$, $\mathbf{C_{i8}}$ is an 8-point integer transform that is numerically **similar to** an 8-point scaled DCT but cannot be produced exactly by scaling and rounding an 8-point DCT matrix, whereas the 4-point integer transform can be produced by scaling and rounding a 4-point DCT matrix.

Table 7.5 8-point Integer Transform Basis C_{f8}

8	8	8	8	8	8	8	8
12	10	6	3	−3	−6	−10	−12
8	4	−4	−8	−8	−4	4	8
10	−3	−12	−6	6	12	3	−10
8	−8	−8	8	8	−8	−8	8
6	−12	3	10	−10	−3	12	−6
4	−8	8	−4	−4	8	−8	4
3	−6	10	−12	12	−10	6	−3

2. The 8 × 8 transform processes have a larger dynamic range than the 4 × 4 processes. The forward scaling factor is $1/2^{22}$ and the inverse scaling factor is $1/2^{8}$. Compare with Figure 7.13 and Figure 7.14.

7.2.4.1 Forward transform $\mathbf{C_{f8}}$: 8 × 8 blocks

The 'core' forward transform basis $\mathbf{C_{f8}}$ is shown in Table 7.5.

Normalise each row through multiplication by $\frac{1}{\sqrt{\sum_j c_{rj}^2}}$, i.e. multiply by $\mathbf{R_{f8}}$:

$1/\sqrt{512}$	$1/\sqrt{512}$	...
$1/\sqrt{578}$	$1/\sqrt{578}$	...
$1/\sqrt{320}$	$1/\sqrt{320}$	...
$1/\sqrt{578}$	$1/\sqrt{578}$	...
$1/\sqrt{512}$	$1/\sqrt{512}$	...
$1/\sqrt{578}$	$1/\sqrt{578}$	...
$1/\sqrt{320}$	$1/\sqrt{320}$	...
$1/\sqrt{578}$	$1/\sqrt{578}$	...

Other columns are identical.

The two-dimensional forward transform is obtained by applying C_{f8} to the rows and the columns of the 8 × 8 input block. Hence the combined scaling matrix $\mathbf{S_{f8}} = \mathbf{R_{f8}} \bullet \mathbf{R_{f8}}^{T} =$

1/512	1/544	$1/128\sqrt{10}$	1/544	...
1/544	1/578	$1/136\sqrt{10}$	1/578	
$1/128\sqrt{10}$	$1/136\sqrt{10}$	1/320	$1/136\sqrt{10}$	
1/544	1/578	$1/136\sqrt{10}$	1/578	
...				...

One quadrant shown, the other four quadrants are identical.

Table 7.6 8-point Integer Transform Basis C_{I8}

1	1	1	1	1	1	1	1
12/8	10/8	6/8	3/8	−3/8	−6/8	−10/8	−12/8
1	1/2	−1/2	−1	−1	−1/2	1/2	1
10/8	−3/8	−12/8	−6/8	6/8	12/8	3/8	−10/8
1	−1	−1	1	1	−1	−1	1
6/8	−12/8	3/8	10/8	−10/8	−3/8	12/8	−6/8
1/2	−1	1	−1/2	−1/2	1	−1	1/2
3/8	−6/8	10/8	−12/8	12/8	−10/8	6/8	−3/8

7.2.4.2 Inverse transform $\mathbf{C_{i8}}$: 8 × 8 blocks

The core inverse transform $\mathbf{C_{i8}}$ is shown in Table 7.6. Note that $\mathbf{C_{i8}} = \mathbf{C_{f8}} / 8$.

Normalize rows:

$\mathbf{R}_{i8} =$

$1/\sqrt{8}$	$1/\sqrt{8}$	...
$8/\sqrt{578}$	$8/\sqrt{578}$	...
$1/\sqrt{5}$	$1/\sqrt{5}$	...
$8/\sqrt{578}$	$8/\sqrt{578}$	...
$1/\sqrt{8}$	$1/\sqrt{8}$	...
$8/\sqrt{578}$	$8/\sqrt{578}$	...
$1/\sqrt{5}$	$1/\sqrt{5}$	...
$8/\sqrt{578}$	$8/\sqrt{578}$	...

$\mathbf{S}_{i8} = \mathbf{R}_{i8} \bullet \mathbf{R}_{i8}^{\mathbf{T}} =$

1/8	2/17	$1/\sqrt{40}$	2/17	...
2/17	32/289	$8/17\sqrt{10}$	32/289	
$1/\sqrt{40}$	$8/17\sqrt{10}$	1/5	$8/17\sqrt{10}$	
2/17	32/289	$8/17\sqrt{10}$	32/289	
...				...

Remaining three quadrants are identical.

7.2.4.3 Inverse quantization and scaling: 8 × 8 blocks

The rescaling matrix $\mathbf{V_{i8}}$, specified in the standard, incorporates transform normalization $\mathbf{S_{i8}}$ and inverse quantization or rescaling. $\mathbf{V_{i8}}$ is approximately related to $\mathbf{S_{i8}}$ and Q_{step}, the quantizer

step size, as follows:

$$\mathbf{V}_{i8} \approx \mathbf{S}_{i8} \cdot 2^8 \cdot Q_{step} \tag{7.25}$$

To ensure consistent decoding behaviour, $\mathbf{V_{i8}}$ is defined by the standard as a function of QP. The following Table lists $\mathbf{V_{i8}}$ for the first six values of QP. Only the top-left quadrant of $\mathbf{V}$ is shown, the remaining three quadrants are identical.

QP	$\mathbf{V_{i8}}$
0	$\begin{bmatrix} 20 & 19 & 25 & 19 & \\ 19 & 18 & 24 & 18 & \\ 25 & 24 & 32 & 24 & \dots \\ 19 & 18 & 24 & 18 & \\ & \dots & & & \dots \end{bmatrix}$
1	$\begin{bmatrix} 22 & 21 & 28 & 21 & \\ 21 & 19 & 26 & 19 & \\ 28 & 26 & 35 & 26 & \dots \\ 21 & 19 & 26 & 19 & \\ & \dots & & & \dots \end{bmatrix}$
2	$\begin{bmatrix} 26 & 24 & 33 & 24 & \\ 24 & 23 & 31 & 23 & \\ 33 & 31 & 42 & 31 & \dots \\ 24 & 23 & 31 & 23 & \\ & \dots & & & \dots \end{bmatrix}$
3	$\begin{bmatrix} 28 & 26 & 35 & 26 & \\ 26 & 25 & 33 & 25 & \\ 35 & 33 & 45 & 33 & \dots \\ 26 & 25 & 33 & 25 & \\ & \dots & & & \dots \end{bmatrix}$
4	$\begin{bmatrix} 32 & 30 & 40 & 30 & \\ 30 & 28 & 38 & 28 & \\ 40 & 38 & 51 & 38 & \dots \\ 30 & 28 & 38 & 28 & \\ & \dots & & & \dots \end{bmatrix}$
5	$\begin{bmatrix} 36 & 34 & 46 & 34 & \\ 34 & 32 & 43 & 32 & \\ 46 & 43 & 58 & 43 & \dots \\ 34 & 32 & 43 & 32 & \\ & \dots & & & \dots \end{bmatrix}$

There are six unique values in each matrix $\mathbf{V_{i8}}$. These are defined as a table $\boldsymbol{v}$ in the standard, for QP = 0 to QP = 5:

QP	v_{m0}	v_{m1}	v_{m2}	v_{m3}	v_{m4}	v_{m5}
0	20	18	32	19	25	24
1	22	19	35	21	28	26
2	26	23	42	24	33	31
3	28	25	45	26	35	33
4	32	28	51	30	40	38
5	36	32	58	34	46	43

The positions $\mathbf{v_{mr}}$ map to the following positions in the matrix $\mathbf{V_{i8}}$:

v_{m0}	v_{m3}	v_{m4}	v_{m3}	v_{m0}	v_{m3}	v_{m4}	v_{m3}
v_{m3}	v_{m1}	v_{m5}	v_{m1}	v_{m3}	v_{m1}	v_{m5}	v_{m1}
v_{m4}	v_{m5}	v_{m2}	v_{m5}	v_{m4}	v_{m5}	v_{m2}	v_{m5}
v_{m3}	v_{m1}	v_{m5}	v_{m1}	v_{m3}	v_{m1}	v_{m5}	v_{m1}
v_{m0}	v_{m3}	v_{m4}	v_{m3}	v_{m0}	v_{m3}	v_{m4}	v_{m3}
v_{m3}	v_{m1}	v_{m5}	v_{m1}	v_{m3}	v_{m1}	v_{m5}	v_{m1}
v_{m4}	v_{m5}	v_{m2}	v_{m5}	v_{m4}	v_{m5}	v_{m2}	v_{m5}
v_{m3}	v_{m1}	v_{m5}	v_{m1}	v_{m3}	v_{m1}	v_{m5}	v_{m1}

For a given QP, $\mathbf{V_{i8}}$ is obtained as:

$$\mathbf{V}_{i8} = \boldsymbol{v}(\mathrm{QP}\%6, \mathrm{n}) \cdot 2^{\mathrm{floor}(\mathrm{QP}/6)} \tag{7.26}$$

Where $\boldsymbol{v}$ (r,n) is row r, column n of $\boldsymbol{v}$. $\mathbf{V_{i8}}$ and hence effective Q_{step} doubles as QP increases by 6.

The effective quantizer step size Q_{step} can be estimated as follows:

7.2.4.4 Forward quantization and scaling : 8 × 8 blocks

The forward quantization and scaling matrix $\mathbf{M_{f8}}$ is derived from $\mathbf{S_{f8}}$, $\mathbf{S_{i8}}$ and $\mathbf{V_{i8}}$ as follows:

$$\mathbf{M}_{f8} = \mathrm{round}\left(\frac{\mathbf{S}_{i8} \bullet \mathbf{S}_{f8} \cdot 2^{30}}{\mathbf{V}_{i8}}\right) \tag{7.27}$$

Table 7.7 Estimated Q_{step} (8 × 8 blocks)

QP	Estimated $Q_{step} = \mathbf{V_{i8}} / (\mathbf{Si} \cdot 2^8)$, element by element division
0	0.625
1	0.6875
2	0.8125
3	0.875
4	1.0
5	1.125

For example, $\mathbf{M_{f8}}$ for QP = 0 is:

$$\begin{bmatrix} 13107 & 12222 & 16777 & 12222 & \\ 12222 & 11428 & 15481 & 11428 & \\ 16777 & 15481 & 20972 & 15481 & \ldots. \\ 12222 & 11428 & 15481 & 11428 & \\ & \ldots. & & & \ldots. \end{bmatrix}$$

Only the top-left quadrant is shown, other quadrants are identical. $\mathbf{M_{f8}}$ arrays for other QP values can be obtained in a similar way as before.

7.2.5 DC transforms

If the macroblock is encoded in 16 × 16 Intra prediction mode, in which the entire 16 × 16 luminance component is predicted from neighbouring pixels, each 4 × 4 residual block is first transformed using the 'core' transform described above. The DC coefficient of each 4 × 4 block is then transformed again using a 4 × 4 Hadamard transform (Figure 7.5):

$$Y_D = \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} & & & \\ & W_D & & \\ & & & \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \right) /2 \tag{7.28}$$

Where $\mathbf{W_D}$ is the block of 4 × 4 DC coefficients and $\mathbf{Y_D}$ is the block after transformation. The output block Y_D is scaled and quantized as described in section 7.2.3.7. Applying this extra transform to the DC coefficients may further compact the data prior to encoding.

The corresponding 4 × 4 DC inverse transform (Figure 7.6) is as follows:

$$W_{QD} = \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} & & & \\ & Z_D & & \\ & & & \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \right) \tag{7.29}$$

Where Z_D is the input and W_{QD} is the output block.

Similarly, the DC coefficients of each transformed 4 × 4 chroma block are grouped and transformed for a second time. If the video format is 4:2:0, there are four 4 × 4 blocks in each chroma coefficient and hence the DC coefficients form a 2 × 2 block (Figure 7.9). A 2 × 2 transform is applied. This is a 2 × 2 Hadamard Transform or 2 × 2 DCT: these transforms are identical for N = 2.

$$Y_D = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} W_D & \\ & \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{7.30}$$

The corresponding inverse transform (Figure 7.10) is given by:

$$W_{QD} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} Z_D & \\ & \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{7.31}$$

For 4:2:2 format video, there are eight 4 × 4 blocks in each chroma component and so a 2 × 4 transform is applied to the DC coefficients:

$$Y_D = \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} W_D & \\ & \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \right) \tag{7.32}$$

With the equivalent inverse transform at the decoder:

$$W_{QD} = \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} Z_D & \\ & \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \right) \tag{7.33}$$

In each case a normalization step is required.

7.2.6 Transform and quantization extensions in the High profiles

Frequency dependent quantization, High profiles

The default quantization process in H.264/AVC applies a uniform quantizer step size to every coefficient in a 4 × 4 or 8 × 8 block. However, it has been shown that the human visual system's sensitivity to quantization artefacts, i.e. distortions due to quantization, can vary with coefficient frequency [ix]. Frequency-dependent quantization is an optional tool, only available in the High profiles, that makes it possible to vary the quantizer step size within a block of coefficients. A typical usage might be to reduce the quantizer step size for low frequency coefficients and increase the step size for high frequencies (Figure 7.16). This may have the effect of improving the subjective quality of the decoded video image, since the more sensitive low-frequency coefficients are quantized less and the less sensitive high frequencies are quantized more.

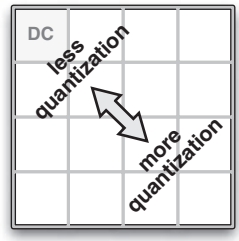


Figure 7.16 Frequency dependent quantization, 4 × 4 block

Table 7.8 Scaling matrices

Scaling list index	Applies to blocks:
0	4 × 4 intra Y
1	4 × 4 intra Cb
2	4 × 4 intra Cr
3	4 × 4 inter Y
4	4 × 4 inter Cb
5	4 × 4 inter Cr
6	8 × 8 intra Y
7	8 × 8 inter Y

Frequency dependent quantization is achieved by transmitting or activating a scaling matrix, a matrix of scaling factors to be applied to the inverse quantization process in the decoder. A flag in the Sequence or Picture Parameter Set indicates whether a scaling matrix is present. If present, one or more scaling matrices are transmitted to the decoder and can then be activated for decoding pictures that refer to the SPS or PPS. Eight scaling matrices are available, each applied to a different block type (Table 7.8).

A specific scaling matrix may be transmitted for each of the eight scaling list indices, or the default scaling matrix may be used. These default scaling matrices are specified in the standard. For example, the default 4 × 4 Intra scaling matrix for Y, Cr or Cb is given by:

6	13	20	28
13	20	28	32
20	28	32	37
28	32	37	42

Recall that 'flat' scaling, i.e. no scaling of the quantizer, is indicated by a scaling factor of 16 (section 7.2.3.6). Hence the 4 × 4 Intra scaling matrix corresponds to the following weighting, after dividing by the 'flat' factor of 16:

0.375	0.8125	1.25	1.75
0.8125	1.25	1.75	2
1.25	1.75	2	2.3125
1.75	2	2.3125	2.625

Hence the DC and two lowest AC coefficients are quantized **less** than the norm and the higher AC coefficients are quantized **more** than the norm.

Lossless predictive coding, High 4:4:4 profiles

This is an alternative to the lossless I_PCM mode described in Chapter 5. A flag in the SPS, qpprime_y_zero_transform_bypass_flag, signals that the rescaling and inverse transform processes are not used, i.e. bypassed. Hence residual samples are directly extracted from the bitstream. With no rescaling and therefore no quantization in the encoder, there is no loss or

distortion introduced by the coding and decoding algorithms. This mode is only available in the High 444, High444Intra and CAVLC444Intra profiles (Chapter 8).

Colour plane coding, High 4:4:4 profiles
A flag in the Sequence Parameter Set indicates that Cr and Cb are each processed in the same way as Y, with independent predictions, motion vectors, etc. Hence each colour component is treated as if it were a separate monochrome Y component. This may be useful for:

(a) parallel processing in the encoder and/or decoder, by handling each colour component independently; and/or
(b) applying the more sophisticated Luma intra prediction tools (Chapter 6) to the chroma components.

7.3 Block scan orders

Blocks of transform coefficients are scanned, i.e. converted to a linear array, prior to entropy coding. The scan order is intended to group together significant coefficients, i.e. non-zero quantized coefficients. In a typical block in a progressive frame, non-zero coefficients tend to be clustered around the top left 'DC' coefficient (Chapter 3). In this case, a zigzag scan order may be the most efficient, shown in Figure 7.17, 4 × 4 and 8 × 8 blocks. After scanning

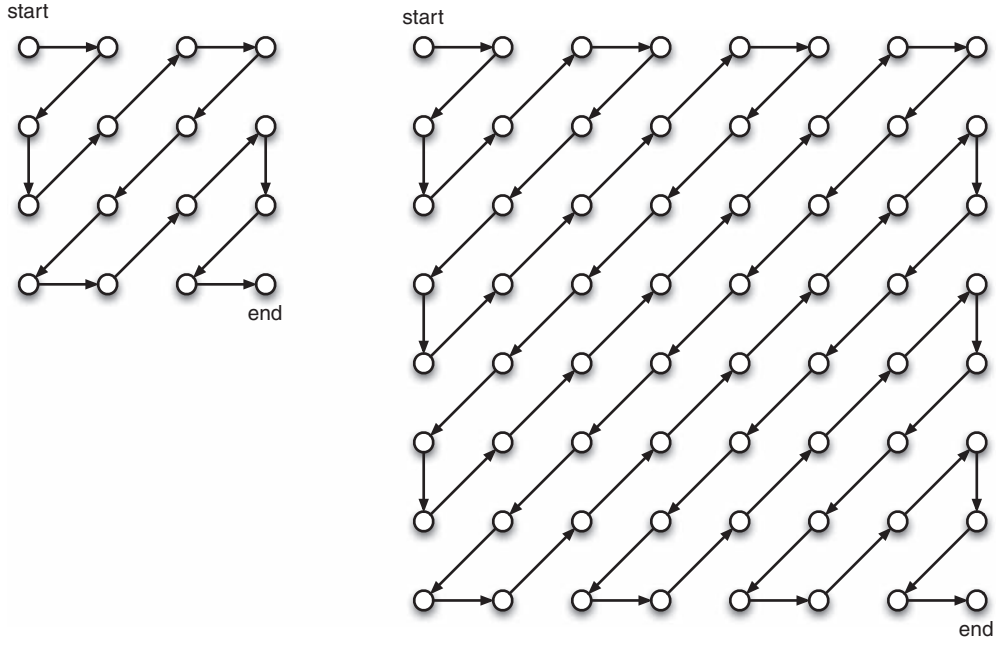


Figure 7.17 Progressive scan orders for 4 × 4 and 8 × 8 blocks

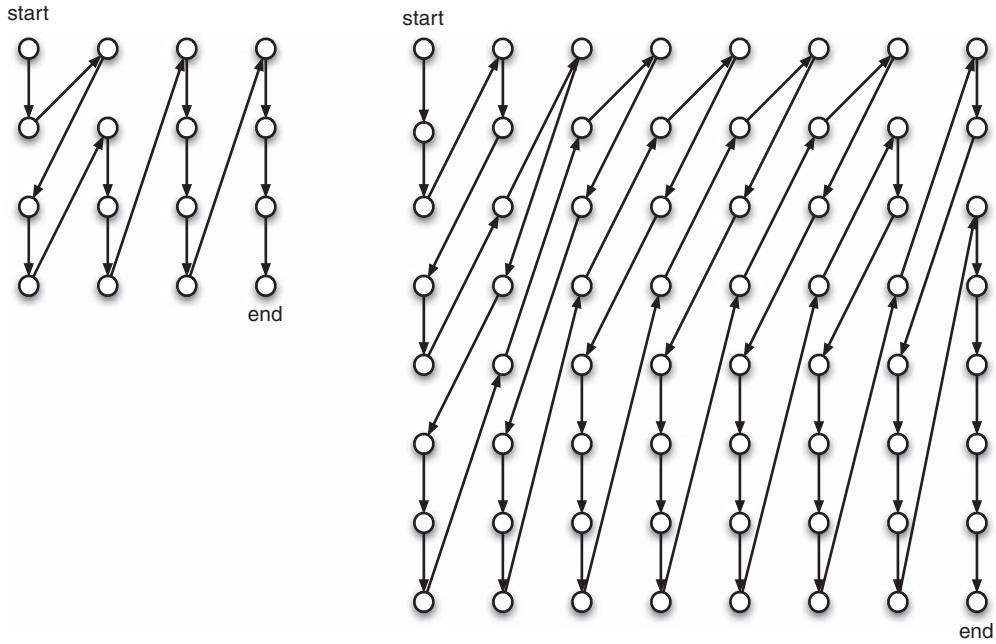


Figure 7.18 Field scan orders for 4 × 4 and 8 × 8 blocks

the block in a zigzag order, the coefficients are placed in a linear array in which most of the non-zero coefficients tend to occur near the start of the array.

However, in an interlaced field or a field of a progressive frame converted from interlaced content, vertical frequencies in each block tend to dominate because the field is vertically sub-sampled from the original scene (Chapter 3). This means that non-zero coefficients tend to occur at the top and towards the left side of the block. A block in a field macroblock is therefore scanned in a modified field scan order (Figure 7.18).

7.4 Coding

A coded H.264 stream or an H.264 file consists of a series of coded symbols. These symbols make up the syntax described in Chapter 5 and include parameters, identifiers and delimiting codes, prediction types, differentially coded motion vectors and transform coefficients. The H.264/AVC standard specifies several methods for coding the symbols, i.e. converting each symbol into a binary pattern that is transmitted or stored as part of the bitstream. These methods are as follows:

Fixed length code: A symbol is converted into a binary code with a specified length (n bits).

Exponential-Golomb variable length code: The symbol is represented as an Exp-Golomb codeword with a varying number of bits (v bits). In general, shorter Exp-Golomb codewords are assigned to symbols that occur more frequently.

CAVLC: Context-Adaptive Variable Length Coding, a specially-designed method of coding transform coefficients in which different sets of variable-length codes are chosen depending on the statistics of recently-coded coefficients, using context adaptation.

CABAC: Context-Adaptive Binary Arithmetic Coding, a method of arithmetic coding in which the probability models are updated based on previous coding statistics.

Symbols occurring in the syntax above the slice data level (Chapter 5) are coded using Fixed Length Codes or Exp-Golomb codes. Symbols at the slice data level and below are coded in one of two ways. If CABAC mode is selected, all of these symbols are coded using CABAC; otherwise, coefficient values are coded using CAVLC and other symbols are coded using fixed length or Exp-Golomb codes.

7.4.1 Exp-Golomb Coding

Exponential Golomb codes, Exp Golomb or ExpG, are binary codes with varying lengths constructed according to a regular pattern [x, xi]. Variable length binary codes such as ExpG codes may be used as an efficient way of representing data symbols with varying probabilities (Chapter 3). By assigning short codewords to frequently-occurring data symbols and long codewords to less common data symbols, the data may be represented in a compressed form.

Table 7.9 lists the first few Exp-Golomb codewords, indexed by a parameter code_num. It is clear from the table that these codes have a regular, logical construction. Exponential-Golomb codes are variable length codes with the folliowing properties:

Table 7.9 Exp-Golomb Codewords

code_num	Codeword
0	1
1	010
2	011
3	00100
4	00101
5	00110
6	00111
7	0001000
8	0001001
...	...

(i) Code length increases with the index code_num and
(ii) Each code can be constructed logically and decoded algorithmically without the need for look-up tables.

An Exp-Golomb codeword has the following structure:

```
[Zero prefix][1][INFO]
```

The codeword consists of a prefix of M zeros, where M is 0 or a positive integer, a 1 and an M-bit information field, INFO. Each codeword may be generated algorithmically from the parameter code_num:

```
M = floor(log_2 [code_num + 1])
INFO = code_num + 1 − 2^M
```

Conversely, code_num may be decoded as follows:

1. Read a series of consecutive zeros until a 1 is detected. Count the number of zeros (M).
2. Read a 1 (ignore).
3. Read M bits = INFO.
4. Code_num = 2^M + INFO – 1.

Note that the length of the codeword is 2M + 1 bits.

Examples:

(a) code_num = 107 :	$\log_2$ [108] = 6.754. . ., M = 6 INFO = 107 + 1 – 2^6 = 44_{10} = 101100_2 Codeword = 0000001101100
(b) codeword = 00000001100011 :	Count leading zeros: M = 7 INFO = 1100011_2 = 99_{10} Code_num = 2^7 + 99 – 1 = 226

Coding a parameter *k* proceeds as follows. *k* is mapped to *code_num* in one of the ways listed in Table 7.10. The value *code_num* is converted to an Exp-Golomb binary codeword (Table 7.9) which is inserted in the H.264 bitstream. Decoding proceeds as follows. The decoder reads M consecutive zeros and calculates the total length of the next Exp-Golomb codeword as 2M + 1 bits. It reads the remaining M + 1 bits and calculates *code_num* which is then mapped to *k*.

Table 7.10 Mappings to code_num

Mapping type	Description
ue	Unsigned direct mapping, code_num = k. Used for macroblock type, reference frame index and others.
te	Truncated mapping: if the largest possible value of k is 1, then a single bit b is sent where b = !code_num, otherwise *ue* mapping is used.
se	Signed mapping, used for motion vector difference, delta QP and others. k is mapped to code_num as follows: $\text{code_num} = 2\lvert k\rvert \quad (k \leq 0)$ $\text{code_num} = 2\lvert k\rvert - 1 \quad (k > 0)$ code_num is mapped to k as follows: $k = (-1)^{\text{code_num}+1}\text{ceil}(\text{code_num} / 2)$
me	Mapped symbols, k is mapped to code_num according to a table specified in the standard.

Example

An inter-coded macroblock in a Baseline Profile bitstream is coded as follows:

Symbol name	Mapping	Notes
mb_type	ue(v)	Macroblock type; unsigned mapping to Exp-Golomb code with variable number of bits.
ref_index_l0	te(v)	Reference picture index, one per macroblock partition; truncated unsigned mapping to Exp-Golomb code.
mvd_l0	se(v)	Motion vector difference, two per macroblock partition; signed mapping to Exp-Golomb code.
coded_block_pattern	me(v)	Identifies 8×8 blocks containing non-zero coefficients; mapping to Exp-Golomb code according to specific table(s) in H.264 standard.
mb_qp_delta	se(v)	Differentially coded quantizer parameter; signed mapping to Exp-Golomb code
Residual...		Residual data coded using CAVLC.

7.4.2 Context Adaptive Variable Length Coding, CAVLC

Context Adaptive Variable Length Coding (CAVLC) is used to encode residual, scan ordered blocks of transform coefficients. CAVLC [xii] is designed to take advantage of several characteristics of quantized coefficient blocks:

1. After prediction, transformation and quantization, blocks are typically sparse, often containing mostly zeros. CAVLC uses run-level coding to compactly represent strings of zeros.

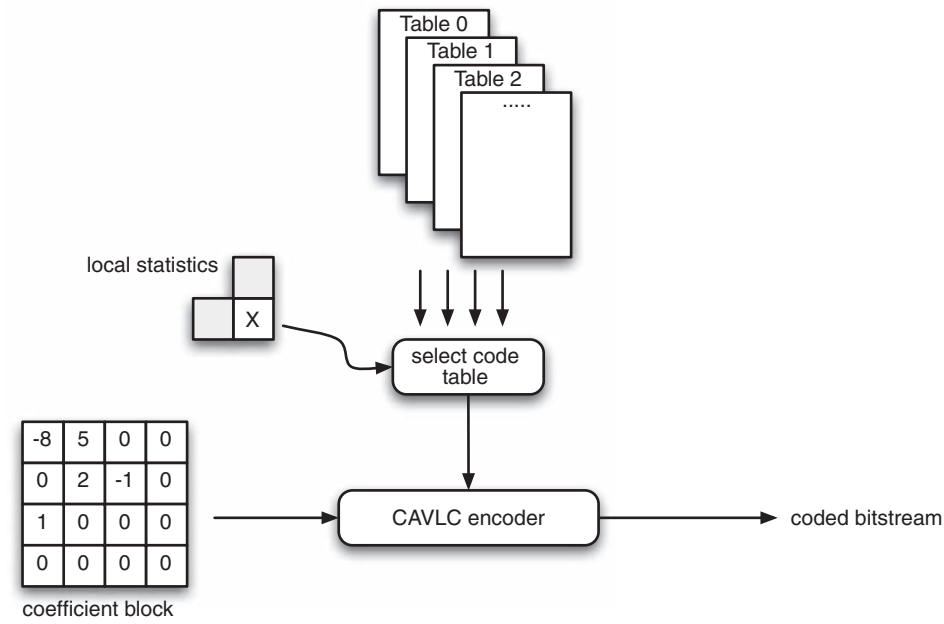


Figure 7.19 CAVLC encoder overview

2. The highest non-zero coefficients after the block scan are often sequences of +/−1 and CAVLC signals the number of high-frequency +/−1 coefficients, 'Trailing 1s' or 'T1s', in a compact way.
3. The number of non-zero coefficients in neighbouring blocks is correlated. The number of coefficients is encoded using a look-up table (coeff_token) and the choice of look-up table depends on the number of non-zero coefficients in neighbouring blocks.
4. The level or magnitude of non-zero coefficients tends to be larger at the start of the scanned array, near the DC coefficient, and smaller towards the higher frequencies. CAVLC takes advantage of this by adapting the choice of VLC look-up table for the level parameter depending on recently coded level magnitudes.

If CAVLC is used together with the 8 × 8 integer transform, each 8 × 8 block of quantized transform coefficients is processed as four 4 × 4 blocks for the purposes of CAVLC encoding and decoding.

Figure 7.19 shows a simplified overview of the CAVLC encoding process. A block of coefficients is scanned using zigzag or field scan and converted into a series of variable length codes (VLCs). Certain VLC tables are chosen based on local statistics, i.e. the number of non-zero coefficients in neighbouring blocks and the magnitude of recently-coded coefficients. CAVLC encoding of a 4 × 4 block of transform coefficients proceeds as follows.

1. Encode the number of coefficients and trailing ones (coeff_token).

The first VLC, coeff_token, encodes both the total number of non-zero coefficients (TotalCoeffs) and the number of trailing +/−1 values (T1). TotalCoeffs can be anything from 0,

Table 7.11 Choice of look-up table for coeff_token

nC	Table for coeff_token
0, 1	VLC table 1
2, 3	VLC table 2
4, 5, 6, 7	VLC table 3
8 or above	FLC

i.e. no coefficients in the 4 × 4 block [2], to 16, i.e. 16 non-zero coefficients, and T1 can take values from 0 to 3. If there are more than three trailing +/−1s, only the last three are treated as 'special cases' and any others are coded as normal coefficients.

For a luma block, there are four choices of look-up table to use for encoding coeff_token, three variable-length code tables and a fixed-length code table. The choice of table depends on the number of non-zero coefficients in the left and upper previously coded blocks, *nA* and *nB* respectively. A parameter *nC* is calculated as follows:

(i) If upper and left blocks are both available, i.e. in the same coded slice, $nC = (nA + nB + 1) \gg 1$, where $\gg$ indicates binary right shift.
(ii) If only the upper is available, $nC = nB$.
(iii) If only the left block is available, $nC = nA$.
(iv) If neither neighbouring block is available, $nC = 0$.

nC selects the look-up table (Table 7.11) so that the choice of VLC adapts to the number of coded coefficients in neighbouring blocks, **context adaptive**. VLC table 1 is biased towards small numbers of coefficients such that low values of TotalCoeffs, 0 and 1, are assigned particularly short codes and high values of TotalCoeff particularly long codes. VLC table 2 is biased towards medium numbers of coefficients, so that TotalCoeff values around 2–4 are assigned relatively short codes, VLC table 3 is biased towards higher numbers of coefficients and a FLC (Fixed Length Code) table assigns a fixed 6-bit code to every value of TotalCoeff.

2. Encode the sign of each T1.

For each trailing +/−1 T1 signalled by coeff_token, the sign is encoded with a single bit, 0 = +, 1 = −, **in reverse order**, starting with the highest-frequency T1.

3. Encode the levels of the remaining non-zero coefficients.

The **level**, i.e. the sign and magnitude, of each remaining non-zero coefficient in the block is encoded **in reverse order**, starting with the highest frequency and working back towards the DC coefficient. The choice of VLC to encode each successive level is context adaptive and depends on the magnitude of the previous coded level.

[2] Note: coded_block_pattern, described in Chapter 5, indicates which **8** × **8** blocks in the macroblock contain non-zero coefficients but, within a coded 8 × 8 block, there may be **4** × **4** sub-blocks that do not contain any coefficients, hence TotalCoeff may be 0 in any 4 × 4 sub-block. In fact, this value of TotalCoeff occurs most often and is assigned the shortest VLC.

Table 7.12 Thresholds for determining whether to increment *suffixLength*

Current *suffixLength*	Threshold to increment *suffixLength*
0	0
1	3
2	6
3	12
4	24
5	48
6	N/A, highest value reached

The level VLC is composed of level_prefix, *b* leading zeros followed by a 1, and level_suffix, an integer code of size *suffixLength* bits. If *suffixLength* is small, the composite VLC, level_prefix + level_suffix, is more efficient for coding small-magnitude coefficients. If *suffixLength* is large, the composite VLC is more efficient for coding large-magnitude coefficients. The choice of *suffixLength* adapts as follows:

1. Start coding the 4×4 luma block. Initialise *suffixLength* to 0, unless there are more than 10 non-zero coefficients and less than 3 trailing ones, in which case initialise *suffixLength* to 1.
2. Encode the highest-frequency non zero coefficient.
3. If the magnitude of this coefficient is larger than a threshold, increment *suffixLength*, up to a maximum *suffixLength* = 6.

In this way, the choice of VLC is matched to the magnitude of the recently-encoded coefficients. Typically, the higher-frequency coefficients, coded first, have smaller magnitudes and the magnitude tends to increase with lower frequency. The level VLC adapts to follow this trend. The thresholds are listed in Table 7.12; the first threshold is zero which means that *suffixLength* is always incremented after the first coefficient level has been encoded.

4. Encode the total number of zeros before the last coefficient.

TotalZeros is the sum of all zeros preceding the highest non-zero coefficient in the reordered array and is coded with a VLC. The reason for sending a separate VLC to indicate TotalZeros is that many blocks contain zero coefficients at the start of the array and, as will be seen later, this approach means that zero-runs at the start of the array need not be encoded.

5. Encode each run of zeros.

The number of zeros preceding each non-zero coefficient (run_before) is encoded in reverse order. A run_before parameter is encoded for each non-zero coefficient, starting with the highest frequency, with two exceptions:

1. If there are no more zeros left to encode, i.e. Σ[run_before] = TotalZeros, it is not necessary to encode any more run_before values.

2. It is not necessary to encode run_before for the final or lowest frequency non-zero coefficient.

The VLC for each run of zeros is chosen depending on (a) the number of zeros that have not yet been encoded (ZerosLeft) and (b) run_before. For example, if there are only two zeros left to encode, run_before can only take 3 values, 0,1 or 2, and so the VLC need not be more than two bits long. If there are six zeros still to encode then run_before can take seven values, 0 to 6, and the VLC table needs to be correspondingly larger.

Example 1

4×4 block:

0	3	−1	0
0	−1	1	0
1	0	0	0
0	0	0	0

Reordered block:
0,3,0,1,−1,−1,0,1,0. . .
TotalCoeffs = 5, indexed from highest frequency, 4, to lowest frequency, 0
TotalZeros = 3
T1s = 3. In fact there are four trailing ones but only three can be encoded as a 'special case'.
Encoding:

Element	Value	Code
coeff_token	TotalCoeffs = 5, T1s = 3 (use Num_VLC0)	0000100
T1 sign (4)	+	0
T1 sign (3)	−	1
T1 sign (2)	−	1
Level (1)	+1 (level_prefix = 1; suffixLength = 0)	1
Level (0)	+3 (level_prefix = 001, suffixLength = 1)	0010
TotalZeros	3	111
run_before(4)	ZerosLeft = 3; run_before = 1	10
run_before(3)	ZerosLeft = 2; run_before = 0	1
run_before(2)	ZerosLeft = 2; run_before = 0	1
run_before(1)	ZerosLeft = 2; run_before = 1	01
run_before(0)	ZerosLeft = 1; run_before = 1	No code required; last coefficient.

The transmitted bitstream for this block is 000010001110010111101101.
Decoding:
The output array is 'built up' from the decoded values as shown below. Values added to the output array at each stage are underlined.

Code	Element	Value	Output array
0000100	coeff_token	TotalCoeffs = 5, T1s = 3	Empty
0	T1 sign	+	$\underline{1}$
1	T1 sign	−	$\underline{-1}$, 1
1	T1 sign	−	$\underline{-1}$, −1, 1
1	Level	+1 (suffixLength = 0; increment suffixLength)	$\underline{1}$, −1, −1, 1
0010	Level	+3 (suffixLength = 1)	$\underline{3}$, 1, −1, −1, 1
111	TotalZeros	3	3, 1, −1, −1, 1
10	run_before	1	3, 1, −1, −1, $\underline{0}$, 1
1	run_before	0	3, 1, −1, −1, 0, 1
1	run_before	0	3, 1, −1, −1, 0, 1
01	run_before	1	3, $\underline{0}$, 1, −1, −1, 0, 1

The decoder has already inserted two zeros, TotalZeros is equal to 3 and so another 1 zero is inserted before the lowest coefficient, making the final output array:

$\underline{0}$, 3, 0, 1, −1, −1, 0, 1

Example 2

4 × 4 block:

−2	4	0	−1
3	0	0	0
−3	0	0	0
0	0	0	0

Reordered block:
−2, 4, 3, −3, 0, 0, −1, . . .
TotalCoeffs = 5, indexed from highest frequency (4) to lowest frequency (0)
TotalZeros = 2
T1s = 1
Encoding:

Element	Value	Code
coeff_token	TotalCoeffs = 5, T1s = 1 (use Num_VLC0)	0000000110
T1 sign (4)	–	1
Level (3)	Sent as –2 (**see note 1**) (suffixLength = 0)	0001
Level (2)	3 (suffixLength = 1)	0010
Level (1)	4 (suffixLength = 1)	00010
Level (0)	−2 (suffixLength = 2)	111
TotalZeros	2	0011
run_before(4)	ZerosLeft = 2; run_before = 2	00
run_before(3..0)	0	No code required

The transmitted bitstream for this block is 0000000110100010010000010111001100.

Note 1: Level (3), with a value of −3, is encoded as a special case. If there are less than 3 T1s, then the first **non**-T1 level cannot have a value of +/−1, otherwise it would have been encoded as a T1. To save bits, this level is incremented if negative, decremented if positive, so that +/−2 maps to +/−1, +/−3 maps to +/−2, and so on. In this way, shorter VLCs are used.

Note 2: After encoding level (3), suffixLength is incremented because the magnitude of this level is greater than the first threshold, which is 0. After encoding level (1), with a magnitude of 4, suffixLength is incremented again because level (1) is greater than the second threshold of 3. Note that the final level (−2) uses a different code from the first encoded level, also −2.

Decoding:

Code	Element	Value	Output array
0000000110	coeff_token	TotalCoeffs = 5, T1s = 1	Empty
1	T1 sign	−	$\underline{-1}$
0001	Level	−2 decoded as −3	$\underline{-3}$, −1
0010	Level	+3	$\underline{+3}$, −3, −1
00010	Level	+4	$\underline{+4}$, 3, −3, −1
111	Level	−2	$\underline{-2}$, 4, 3, −3, −1
0011	TotalZeros	2	−2, 4, 3, −3, −1
00	run_before	2	−2, 4, 3, −3, $\underline{0, 0}$, −1

All zeros have now been decoded and so the output array is:
−2, 4, 3, −3, 0, 0, −1
This example illustrates how bits are saved by encoding TotalZeros: only a single zero run needs to be coded even though there are five non-zero coefficients.

Example 3

4 × 4 block:

0	0	1	0
0	0	0	0
1	0	0	0
−1	0	0	0

Reordered block:
0,0,0,1,0,1,0,0,0,−1
TotalCoeffs = 3, indexed from highest frequency (2) to lowest frequency (0)
TotalZeros = 7
T1s = 3

Encoding:

Element	Value	Code
coeff_token	TotalCoeffs = 3, T1s = 3 (use Num_VLC0)	00011
T1 sign (2)	−	1
T1 sign (1)	+	0
T1 sign (0)	+	0
TotalZeros	7	011
run_before(2)	ZerosLeft = 7; run_before = 3	100
run_before(1)	ZerosLeft = 4; run_before = 1	10
run_before(0)	ZerosLeft = 3; run_before = 3	No code required; last coefficient.

The transmitted bitstream for this block is 0001110001110010.
Decoding:

Code	Element	Value	Output array
00011	coeff_token	TotalCoeffs = 3, T1s = 3	Empty
1	T1 sign	−	$\underline{-1}$
0	T1 sign	+	$\underline{1}, -1$
0	T1 sign	+	$\underline{1}, 1, -1$
011	TotalZeros	7	$1, 1, -1$
100	run_before	3	$1, 1, 0, 0, 0, -1$
10	run_before	1	$1, 0, 1, 0, 0, 0, -1$

The decoder has inserted four zeros. TotalZeros is equal to 7 and so another three zeros are inserted before the lowest coefficient:
$0, 0, 0, 1, 0, 1, 0, 0, 0, -1$

7.4.3 Context Adaptive Binary Arithmetic Coding, CABAC

Context-based Adaptive Binary Arithmetic Coding (CABAC) [xiii, xiv] is an optional entropy coding mode available in Main and High profiles. CABAC achieves good compression performance through:

(a) selecting probability models for each syntax element according to the element's context,
(b) adapting probability estimates based on local statistics and
(c) using arithmetic coding rather than variable-length coding.

Coding a data symbol involves the following stages.

1. Binarization: CABAC uses Binary Arithmetic Coding which means that only binary decisions (1 or 0) are encoded. A non-binary-valued symbol, e.g. a transform coefficient or motion vector, is 'binarized' or converted into a binary code prior to arithmetic coding. This process is similar to the process of converting a data symbol into a variable length

code (section 7.4.1) but the binary code is further encoded by the arithmetic coder prior to transmission.

Stages 2, 3 and 4 are repeated for each bit or 'bin' of the binarized symbol:

2. Context model selection. A 'context model' is a probability model for one or more bins of the binarized symbol and is chosen from a selection of available models depending on the statistics of recently-coded data symbols. The context model stores the probability of each bin being '1' or '0'.
3. Arithmetic encoding: An arithmetic coder encodes each bin according to the selected probability model (Chapter 3). Note that there are just two sub-ranges for each bin, corresponding to the possible values of '0' and '1'.
4. Probability update: The selected context model is updated based on the actual coded value. E.g. if the bin value was '1', the frequency count of '1's is increased.

The coding process

The CABAC coding process is illustrated in Figure 7.20. A syntax element is binarized, converted to a series of bits, each of which corresponds to a single binary decision or **bin**. If the probability of the bin contents being 1 or 0 is likely to remain at 0.5, e.g. for a +/− sign bit, a simple bypass coding process occurs. Otherwise, the probability of the bin containing a 1 or a 0 is modelled based on (a) previous counts of 1 or 0 and (b) values of the same syntax element in the immediate neighbourhood. The chosen probabilities are passed to an arithmetic coder which codes the bin. The context model is updated based on the actual bin contents (1 or 0).

We will illustrate the coding process for one example, MVD_x, motion vector difference in the x-direction.

1. Binarize the value MVD_x. MVD_x is mapped to the following table (Table 7.13) of uniquely-decodeable codewords for $|MVD_x|<9$. Larger values of MVD_x are binarized using an Exp-Golomb codeword.
   The first bit of the binarized codeword is bin 1, the second bit is bin 2 and so on.

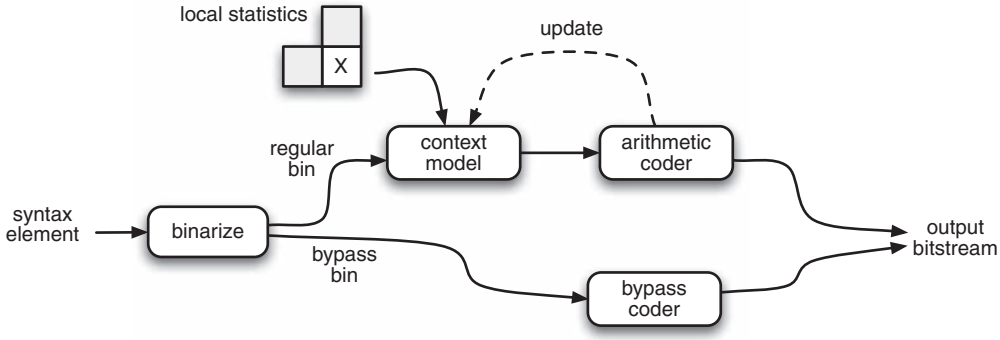


Figure 7.20 CABAC coding process overview

Table 7.13 Binarization of MVD magnitude

$\lvert MVD_x \rvert$	Binarization								
0	0								
1	1	0							
2	1	1	0						
3	1	1	1	0					
4	1	1	1	1	0				
5	1	1	1	1	1	0			
6	1	1	1	1	1	1	0		
7	1	1	1	1	1	1	1	0	
8	1	1	1	1	1	1	1	1	0
Bin number	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**

2. Choose a **context model** for each bin. One of three models is selected for bin 1 (Table 7.14), based the L1 norm of two previously-coded MVD values, e_k:

   $e_k = \lvert MVD_A \rvert + \lvert MVD_B \rvert$ where A and B are the blocks immediately to the left and above the current block respectively.

   If e_k is small, then there is a high probability that the current MVD will have a small magnitude and conversely, if e_k is large then it is more likely that the current MVD will have a large magnitude. A probability table or context model is selected accordingly. The remaining bins are coded using one of four further context models (Table 7.15).
3. Encode each bin. The selected context model supplies two probability estimates, the probability that the bin contains '1' and the probability that the bin contains '0', that determine the two sub-ranges used by the arithmetic coder to encode the bin. The sign of MVD (+ or −) is coded using the Bypass routine, assuming that + and – are equally probable.
4. Update the context models. For example, if context model 2 is selected for bin 1 and the value of bin 1 is '0', the frequency count of '0's is incremented so that the next time this model is selected, the probability of an '0' will be slightly higher. When the total number of occurrences of a model exceeds a threshold value, the frequency counts for '0' and '1' will be scaled down, which in effect gives higher priority to recent observations.

Table 7.14 Context models for bin 1

e_k	Context model for bin 1
$0 \le e_k < 3$	Model 0
$3 \le e_k < 33$	Model 1
$33 \le e_k$	Model 2

Table 7.15 Context models

Bin	Context model
1	0, 1 or 2 depending on e_k
2	3
3	4
4	5
5	6
6 and higher	6

The context models

Context models and binarization schemes for each syntax element are defined in the standard. There are nearly 300 separate context models for the various syntax elements. Some models have different uses depending on the slice type, for example, skipped macroblocks are not permitted in an I-slice and so context models 0-2 are used to code bins of mb_skip or mb_type depending on whether the current slice is Intra coded. At the beginning of each coded slice, the context models are initialised depending on the initial value of the Quantization Parameter QP, since this has a significant effect on the probability of occurrence of the various data symbols.

The arithmetic coding engine

The arithmetic decoder is described in detail in the Standard and has three distinct properties:

1. Probability estimation is performed by a transition process between 64 separate probability states for 'Least Probable Symbol', LPS, the least probable of the two binary decisions '0' or '1'.
2. The range R representing the current state of the arithmetic coder (Chapter 3) is quantized to a small range of pre-set values before calculating the new range at each step, making it possible to calculate the new range using a look-up table, without the use of multiplications.
3. A simplified encoding and decoding process is defined for data symbols with a near-uniform probability distribution.

The definition of the decoding process is designed to facilitate low-complexity implementations of arithmetic encoding and decoding. Overall, CABAC can provide improved coding efficiency compared with CAVLC at the expense of greater computational complexity on some processing platforms. Chapter 9 compares the performance of CAVLC and CABAC.

7.5 Summary

After intra or inter prediction, blocks of residual data in a macroblock are transformed, quantized, scanned and coded. An H.264/AVC codec uses a number of transforms that are based on the well known DCT and Hadamard Transforms but with certain novel aspects. In contrast with previous standards, each inverse transform is exactly specified using integer

arithmetic. Efficient implementation of the transform and quantization is made possible by 'absorbing' part of the transform into the quantizer process.

An H.264 encoder converts a video signal into a set of quantized transform coefficients, motion vectors, prediction parameters and header parameters. These values are coded to produce a compressed bitstream. An encoder may choose between variable-length coding, using Exponential Golomb codes for some syntax elements and a custom-designed Context Adaptive Variable Length Coding algorithm for residual blocks, or a Context Adaptive Binary Arithmetic Coder.

7.6 References

i. ISO/IEC 10918-1:1994, 'Information technology – Digital compression and coding of continuous-tone still images: Requirements and guidelines', JPEG, 1994.
ii. ISO/IEC 13818-2:2000, 'Information technology – Generic coding of moving pictures and associated audio information: Video', MPEG-2 Video, 2000.
iii. ISO/IEC 14496-2:2004, 'Information technology – Coding of audio-visual objects – part 2: Visual', MPEG-4 Visual, 2004.
iv. IEEE Std 1180-1990 (withdrawn), 'IEEE Standard Specifications for the Implementations of 8 × 8 Inverse Discrete Cosine Transform', 1990.
v. SMPTE 421M-2006, 'VC-1 Compressed Video Bitstream Format and Decoding Process', 2006.
vi. A. Hallapuro, M. Karczewicz and H. Malvar, 'Low Complexity Transform and Quantization – Part I: Basic Implementation', ITU-T SG16 Q.6 and ISO/IEC JTC/SC29/WG11 document JVT-B038, Geneva, 2002.
vii. H. S. Malvar, A. Hallapuro, M. Karczewicz and L. Kerofsky, 'Low-complexity transform and quantization in H.264/AVC', *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7 (2003), pp. 598–603.
viii. S. Gordon, D. Marpe and T. Wiegand, 'Simplified Use of 8 × 8 Transforms – Proposal', ITU-T SG16 Q.6 and ISO/IEC JTC/SC29/WG11 document JVT-J029, Hawaii, 2003.
ix. I Hontsch, L Karam, 'Locally adaptive perceptual image coding', *IEEE Transactions on Image Processing*, vol. 9, no. 9, pp. 1472–1483, 2000.
x. S. W. Golomb, 'Run-length encoding', *IEEE Transactions on Information Theory*, vol. IT-12, pp. 399–401, 1966.
xi. J. Teuhola, 'A compression method for clustered bit-vectors', *Information Processing Letters*, vol. 7, pp. 308–311, October 1978.
xii. G. Bjøntegaard and K. Lillevold, 'Context-adaptive VLC coding of coefficients', JVT document JVT-C028, Fairfax, May 2002.
xiii. D. Marpe, G. Blättermann and T. Wiegand, 'Adaptive Codes for H.26L', ITU-T SG16/6 document VCEG-L13, Eibsee, Germany, January 2001.
xiv. D. Marpe, H. Schwarz, and T. Wiegand, 'Context-Based Adaptive Binary Arithmetic Coding in the H.264 / AVC Video Compression Standard', *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 620–636, July 2003.

8

H.264 conformance, transport and licensing

8.1 Introduction

Chapters 4,5,6 and 7 covered the basic concepts of H.264 Advanced Video Compression and the key algorithms that enable an H.264 codec to efficiently code and decode video. The purpose of an industry standard is to enable interoperability between encoders, bitstreams and decoders; i.e. the standard makes it possible for a bitstream encoded by one manufacturer's encoder to be decoded by a different manufacturer's decoder. H.264/AVC defines Profiles and Levels to place operational limits on (a) the particular coding tools and (b) the computational capacity and storage required to decode a sequence. Conformance is verified using a theoretical 'model', a Hypothetical Reference Decoder.

Practical applications of H.264/AVC involve transmitting and/or storing coded video information. The standard includes a number of features designed to support efficient, robust transport of the coded bitstream, including Parameter Sets and NAL Units, described in Chapter 5, and specific transport tools, described in this chapter. To help support an increasingly diverse range of video content and display types, 'side' information including Supplemental Enhancement Information and Video Usability Information may be transmitted along with the coded video data.

Video coding is big business, with many worldwide industries relying on video compression to enable digital media products and services. With many thousands of published patents in the field of video coding, licensing of H.264/AVC implementations is an important issue for commercial digital video applications.

8.2 Conforming to the Standard

H.264/AVC specifies many syntax options and decoding algorithms [i] which cover a wide range of potential video coding scenarios. The standard is designed to support video coding for applications such as small hand-held devices with limited display resolution and minimal computational capacity, through to high-definition decoders with large amounts of memory

The H.264 Advanced Video Compression Standard, Second Edition Iain E. Richardson
© 2010 John Wiley & Sons, Ltd

and computing resources. The standard describes the various syntax elements that may occur in a bitstream and specifies exactly how each syntax element should be processed and decoded in order to produce an output video sequence.

It is important to know whether a particular decoder can handle a particular coded sequence, i.e. whether the decoding and display operations are within the decoder's capabilities. This is achieved by specifying a **profile** and **level** for every coded sequence. The profile places algorithmic constraints on the decoder, determining which decoding tools the decoder should be capable of handling, whilst the level places data processing and storage constraints on the decoder, determining how much data the decoder should be capable of storing, processing and outputting to a display. An H.264 decoder can immediately determine whether it is capable of decoding a particular bitstream by extracting the Profile and Level parameters and determining whether these are supported by the decoder's capabilities.

8.2.1 Profiles

The H.264/AVC standard specifies a number of **Profiles**, each specifying a subset of the coding tools available in the H.264 standard. A Profile places limits on the algorithmic capabilities required of an H.264 decoder. Hence a decoder conforming to the Main Profile of H.264 only needs to support the tools contained within the Main Profile; a High Profile decoder needs to support further coding tools; and so on. Each Profile is intended to be useful to a class of applications. For example, the Baseline Profile may be useful for low-delay, 'conversational' applications such as video conferencing, with relatively low computational requirements. The Main Profile may be suitable for basic television/entertainment applications such as Standard Definition TV services. The High Profiles add tools to the Main Profile which can improve compression efficiency especially for higher spatial resolution services, e.g. High Definition TV.

8.2.1.1 Baseline, Constrained Baseline, Extended and Main Profiles

Figure 8.1 shows the tools supported by the Baseline, Constrained Baseline, Extended and Main Profiles. The **Baseline Profile** was originally intended to be suitable for low complexity, low delay applications such as conversational or mobile video transmission. It includes I and P slice types, allowing intra prediction and motion compensated prediction from a single reference, the basic 4×4 integer transform and CAVLC entropy coding. It also supports three tools for improved transport efficiency, FMO, ASO and Redundant Slices (section 8.3). However, these last three tools have not tended to be popular with codec manufacturers and most implementations of H.264/AVC do not support FMO, ASO or Redundant Slices. In recognition of this, a recent amendment to the standard includes the **Constrained Baseline Profile** which excludes these tools [ii].

The **Extended Profile** is a superset of the Baseline Profile, adding further tools that may be useful for efficient network streaming of H.264 data (section 8.3). The **Main Profile** is a superset of the Constrained Baseline Profile and adds coding tools that may be suitable for broadcast and entertainment applications such as digital TV and DVD playback, namely CABAC entropy coding and bipredicted B slices with prediction modes such as Weighted Prediction for better coding efficiency and frame/field coding support for interlaced video content.

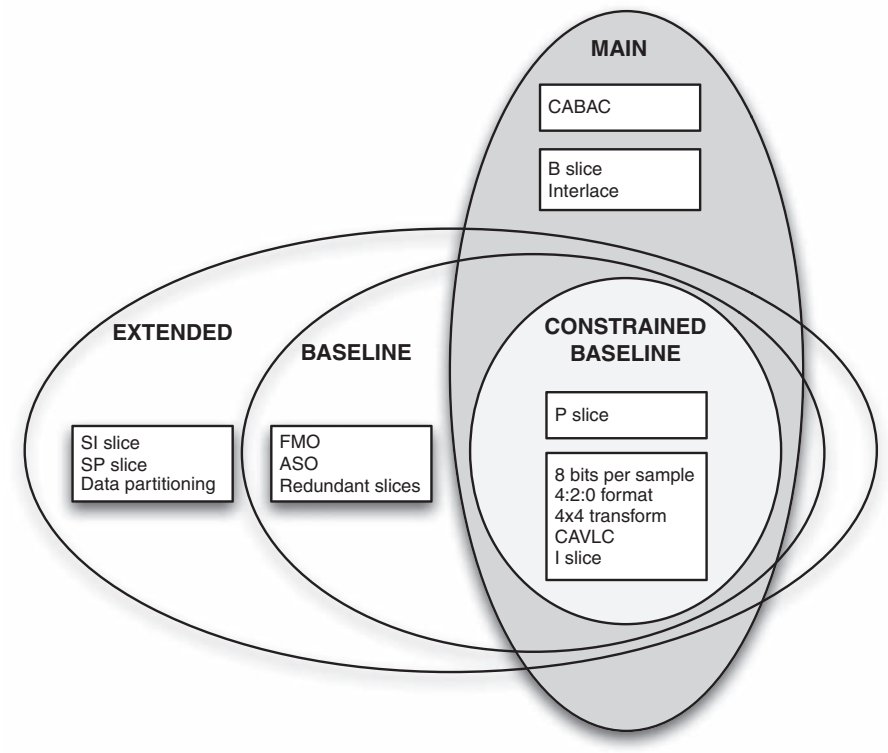


Figure 8.1 Baseline, Constrained Baseline, Extended and Main Profiles

8.2.1.2 High Profiles

Four High Profiles are shown in Figure 8.2, together with the Main Profile for comparison. Each of these Profiles adds coding tools that support higher-quality applications – High Definition, extended bit depths, higher colour depths – at the expense of greater decoding complexity. The High Profile is a superset of the Main Profile and adds the following tools: 8×8 transform and 8×8 inter prediction for better coding performance, especially at higher spatial resolutions, quantizer scale matrices which support frequency-dependent quantizer weightings, separate quantizer parameters for Cr and Cb and support for monochrome video (4:0:0 format). The High Profile makes it possible to use a higher coded data rate for the same Level (see section 8.2.2). The High Profile may be particularly useful for High Definition applications.

Further profiles add more sophisticated tools that may be necessary or useful for 'professional' applications such as content distribution, archiving, etc. The maximum number of bits per sample is extended to 10 bits in the High10 profile and to 14 bits in the High444Pred profile. High422 Profile adds support for 4:2:2 video, i.e. higher Chroma resolution, and High444 Profile extends this to 4:4:4 video giving equal resolution in Luma and Chroma components and adds separate coding for each colour component and a further lossless coding mode that uses predictive coding (Chapter 7).

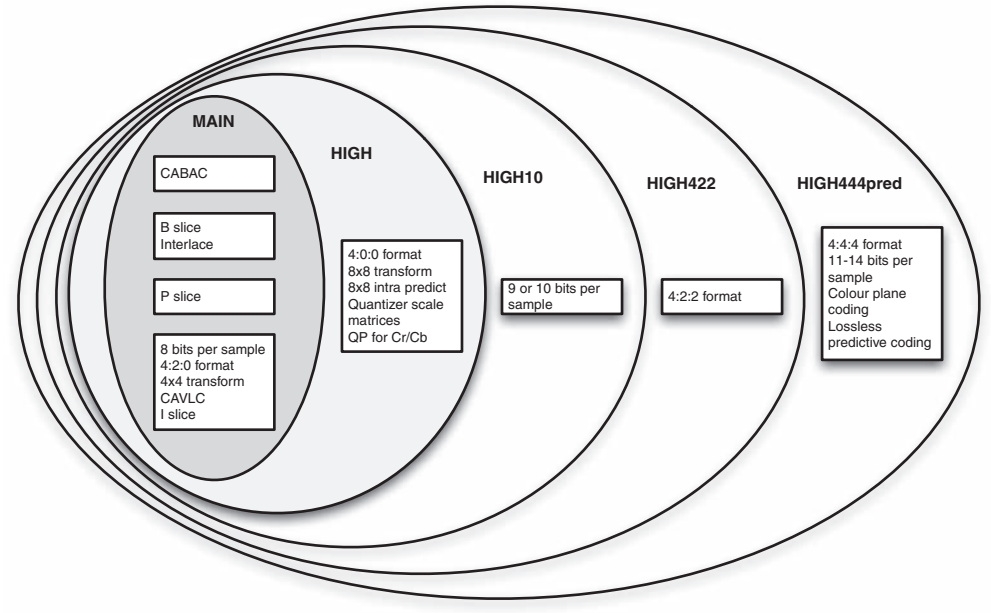


Figure 8.2 Main and High Profiles

8.2.1.3 Intra Profiles

Figure 8.3 shows the Main Profile together with four Intra Profiles. Each of these includes selected tools contained in the High Profiles of Figure 8.2, but without Inter coding support, i.e. no P or B slices, or Interlace support. These Intra Profiles may be useful for applications such as video editing which require efficient coding of individual frames but also require complete random access to coded frames and hence do not require inter coding.

8.2.2 Levels

The Sequence Parameter Set defines a Level for the coded bitstream, a set of constraints imposed on values of the syntax elements in the H.264/AVC bitstream. The combination of Profile and Level constrains the maximum computational and memory requirements that will be placed on the decoder. The main Level constraints are as follows:

- Maximum macroblock processing rate (MaxMBPS): the maximum number of macroblocks, 16×16 luma and associated chroma, that a decoder must handle per second.
- Maximum frame size (MaxFS): the maximum number of macroblocks in a decoded frame.
- Maximum Decoded Picture Buffer size (MaxDPB): the maximum memory space required to store decoded pictures at the decoder.
- Maximum video bit rate (MaxBR): the maximum coded video bitrate.
- Maximum Coded Picture Buffer size (MaxCBP): the maximum memory space required to store (buffer) coded data prior to decoding.

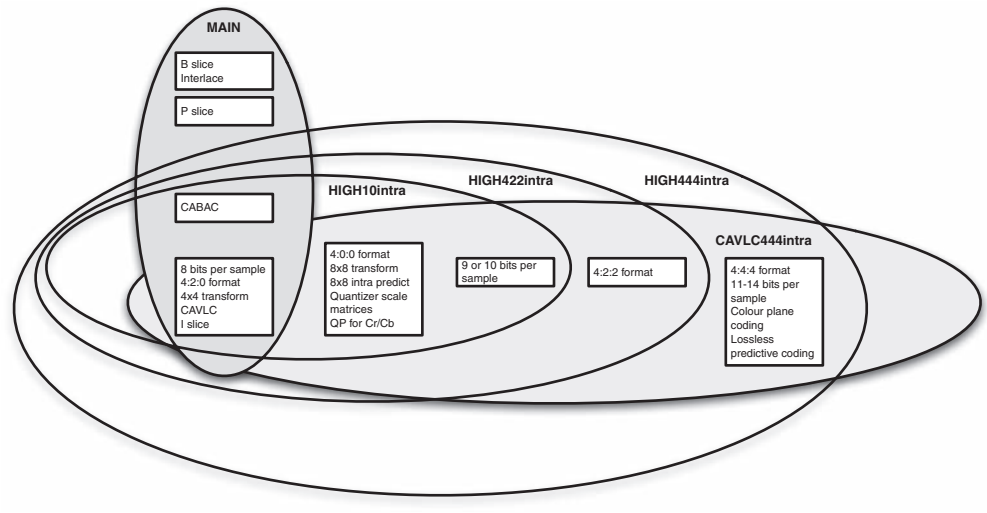


Figure 8.3 Main and Intra Profiles

- Vertical motion vector range (MaxVmvR): the maximum range (+/−) of a vertical motion vector.
- Minimum Compression Ratio (MinCR): the minimum ratio between uncompressed video frames and compressed or coded data size.
- Maximum motion vectors per two consecutive macroblocks (MaxMvsPer2Mb): specified for levels above 3, a constraint on the number of motion vectors (MVx, MVy) that may occur in any two consecutive decoded macroblocks.

In the present version of the standard [i] level numbers range from 1 to 5 with intermediate steps 1.1, 1.2, 1.3, 2.1, etc. A decoder operating at a particular level is expected to be able to handle any of the level constraints at or below that level. For example, a Level 2.1 decoder can handle levels 1, 1.1, 1.2, 1.3, 2 and 2.1. Selected level constraints are shown graphically in Figure 8.4. It is clear that these range from very low, suitable for low-complexity decoders with limited display resolutions, e.g. handheld devices, to very high, suitable for Full High Definition decoders with high resolution displays and significant processing resources.

The parameter MaxFS defines the maximum decoded picture size in macroblocks. This implies certain maximum display resolutions, depending on the aspect ratio. Figure 8.5 shows some examples. For example, MaxFS at Level 1 is equal to 99 macroblocks, which can correspond to 11 × 9 MB or 176 × 144 luma samples, i.e. QCIF resolution. At Levels 2.2 and 3, MaxFS is 1620 macroblocks which can correspond to 45 × 36 MB or 720 × 576 luma samples, '625' Standard Definition. At Levels 4 and 4.1, MaxFS is 8192 macroblocks, which corresponds to approximately 120 × 68 MB or 1920 × 1080 luma samples, '1080p' High Definition. Note that many other aspect ratios are possible within these constraints.

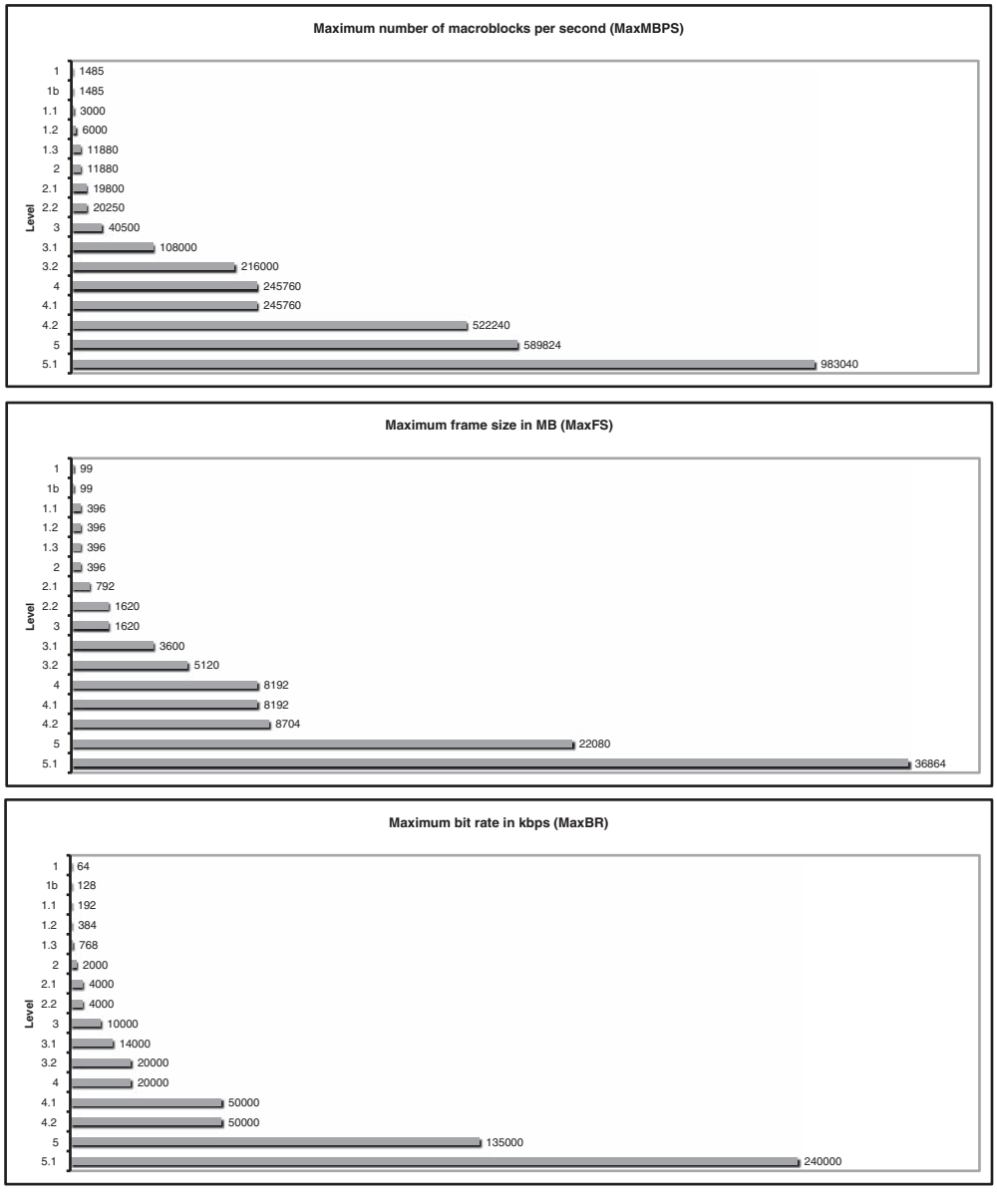


Figure 8.4 Selected Level constraints

The combination of MaxFS, the frame size in macroblocks, and MaxMBPS, the number of macroblocks per second, places a constraint on the maximum frame rate at a particular frame resolution. Table 8.1 lists some examples. Level 1.2 corresponds to CIF resolution at a maximum of 15 frames per second **or** QCIF at a maximum of 60 frames per second. Level 4 corresponds to 1080p High Definition at a maximum of 30 frames per second, **or** 720p High Definition at a maximum of 68 frames per second, and so on.

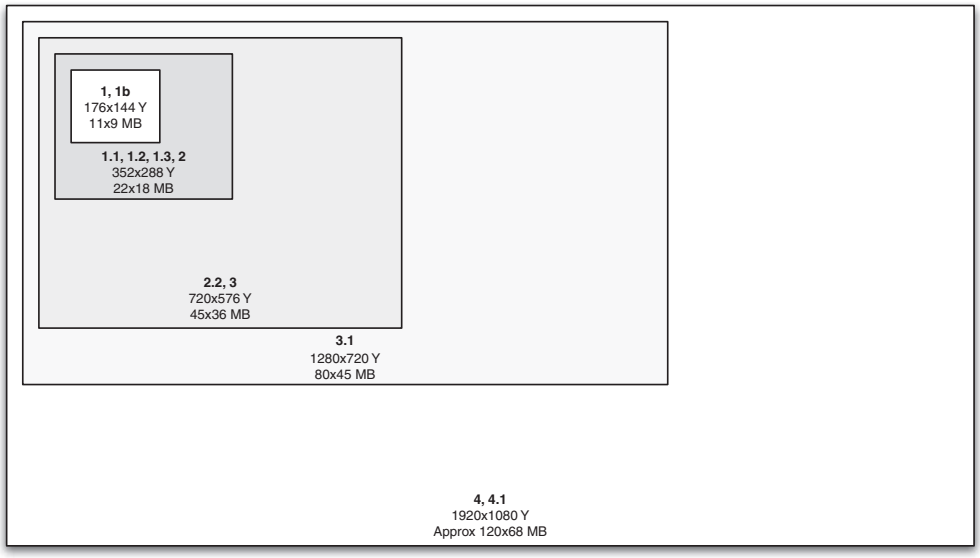


Figure 8.5 Selected display resolutions

Table 8.1 Selected formats, frame rates and levels

Format (luma resolution)	Max frames per second	Level
QCIF (176x144)	15	1, 1b
	30	1.1
CIF (352x288)	15	1.2
	30	1.3, 2
525 SD (720x480)	30	3
625 SD (720x576)	25	3
720p HD (1280x720)	30	3.1
1080p HD (1920x1080)	30	4, 4.1
	60	4.2
4Kx2K (4096x2048)	30	5.1

Figure 8.6 H.264 encoder and decoder buffers

8.2.3 Hypothetical Reference Decoder

As well as ensuring that a decoder can handle the syntax elements and sequence parameters in an H.264 stream, it is important to make sure that the coded sequence 'fits' within the limitations of the decoder buffering and processing capacity. This is handled by defining a Hypothetical Reference Decoder (HRD), a virtual buffering algorithm that can be used to test the behaviour of the coded bitstream and its effect on a real decoder. Annex C of the H.264 standard specifies the Hypothetical Reference Decoder [iii].

Figure 8.6 shows a typical H.264 codec. Video frames are encoded to produce an H.264 bitstream which is buffered prior to transmission. When a frame *n* is coded, the encoder buffer is filled with b_n coded bits. The encoder buffer is emptied at the rate of the transmission channel, r_c bits per second. The dual situation occurs at the decoder, where bits arrive from the channel and fill the Coded Picture Buffer (CPB) at a rate of r_c bits per second. The decoder decodes frame *n*, removing b_n bits from the CPB, and places decoded frames in the Decoded Picture Buffer (DPB). These are then output – displayed – and/or used for prediction to decode further frames.

The HRD (Figure 8.7) is a model of the decoding side of Figure 8.6. In this conceptual model, the H.264 bitstream is output by a Hypothetical Stream Scheduler (HSS) at a constant or varying channel rate into the CPB. Access units, coded pictures, are removed from the CPB and decoded instantaneously, i.e. they are assumed to be instantly decoded and placed in the DPB.

The H.264/AVC standard specifies two types of HRD conformance, one for essential Video Coding Layer (VCL) units and a second for all video coding elements in the stream . In most scenarios a compliant decoder must satisfy both types. The following conditions must be met (among others, simplified from the conditions in the standard):

1. The CPB must never overflow, i.e. the contents must not exceed the maximum CPB size.
2. The CPB must never underflow, i.e. the contents must not reach zero.
3. The DPB must never exceed its maximum size.

The maximum size of the CPB and DPB are specified as part of the Level limits and so the HRD provides a mechanism for checking and enforcing the Level constraints. The operation of the HRD can be illustrated with some examples.

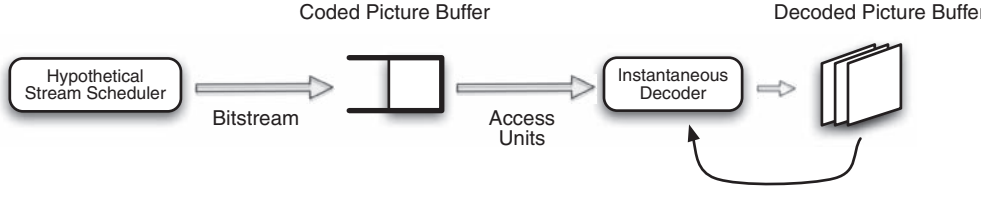


Figure 8.7 Hypothetical Reference Decoder (HRD)

Example 1: Typical HRD operation

Video frame rate:	5 frames per second
Channel bit rate:	5000 bits per second : constant bit rate
Initial removal delay:	0.8 seconds : see below
Maximum CPB size:	6000 bits

The bitstream consists of a series of access units with the following coded sizes (Table 8.2).

Figure 8.8 shows the behaviour of the encoder output buffer. Frame 0 is encoded and added to the buffer at time 0 and each subsequent frame is added at intervals of 0.2 seconds. At the same time, the channel empties the buffer at a constant rate of 5000 bits per second. Frames larger than (bitrate/frame rate) = 1000 bits cause the buffer to fill up; frames smaller than 1000 bits cause the buffer to empty.

The encoder buffer behaves like a 'leaky bucket' : filling at a variable rate depending on the coded size of each access unit and emptying or leaking at a constant rate, the bitrate of the channel.

The corresponding decoder CPB behaviour is shown in Figure 8.9. The initial CPB removal delay is necessary to allow enough data to be received before frames are decoded, 0.8 seconds in this example. The CPB fills at the constant channel rate during this initial delay period before frame 0 is decoded and instantly removed from the buffer. As the first five frames are decoded, the CPB comes close to the underflow condition. Referring back to Figure 8.8, this

Table 8.2 HRD example 1: access unit sizes

Frame	Coded size (bits)
0	3000
1	1200
2	1200
3	1200
4	1200
5	500
6	500
7	500
8	500
9	1000
10	1000
11	1000
12	1000
13	1000
14	400
15	400
16	400
17	1500
18	1500
19	1500
20	1500
21	1500

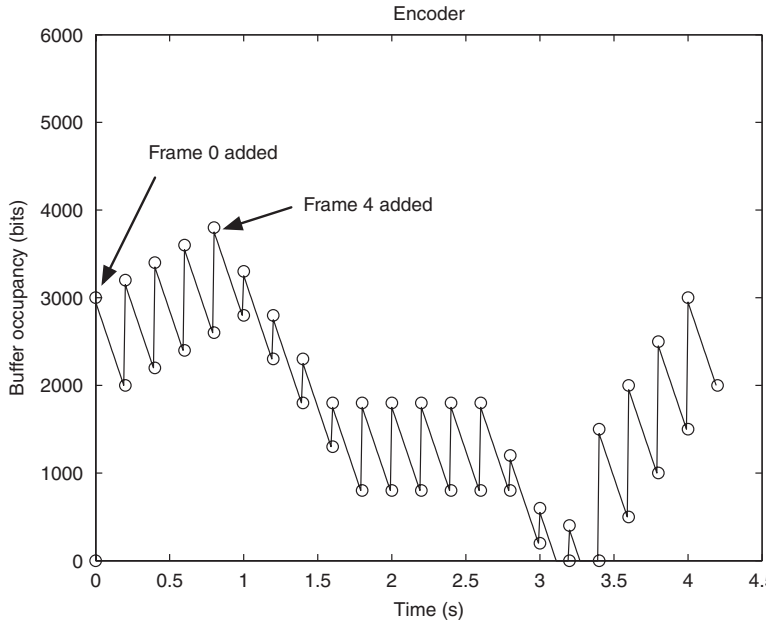


Figure 8.8 HRD example 1: encoder buffer

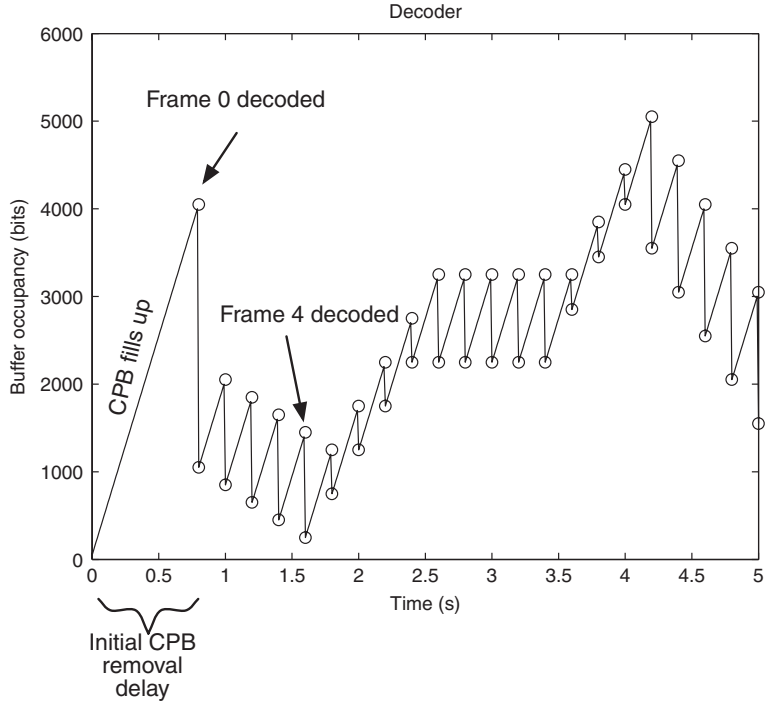


Figure 8.9 HRD example 1: decoder CPB

Table 8.3 HRD example 2: frame sizes

Frame	Coded size (bits)
. . ..	. . ..
10	1000
11	1000
12	1000
13	1000
14	1500
15	1500
16	1500
17	1500
18	1500
19	1500
20	1500
21	1500

corresponds to the initial increasing buffer level at the encoder. In fact, should the encoder buffer occupancy exceed (CPB initial removal delay * bitrate) = 4000 bits, the CPB will underflow. The CPB exhibits the opposite behaviour to the encoder buffer – large frames (>1000 bits) cause the CPB level to decrease, small frames (<1000 bits) cause it to increase.

Example 2: Frame Sizes: CPB underflow

Parameters: Same initial parameters as Example 1.

Frame sizes: Same initial frame sizes, larger frames from frame 14 onwards (Table 8.3).

In the latter part of the sequence, the encoder buffer level exceeds (CPB initial removal delay * bitrate) = 4000 bits (Figure 8.10) after frame 18 is coded. When frame 18 is removed from the CPB (Figure 8.11), the CPB level is below 0 and the HRD has violated the underflow condition. This particular bitstream is therefore not a conforming H.264/AVC bitstream.

Example 3: Frame Sizes: CPB overflow

Parameters: Same initial parameters as Example 1.

Frame sizes: Same initial frame sizes, smaller frames from frame 14 onwards (Table 8.4).

The encoder buffer is consistently 'drained' to zero (Figure 8.12) due to the series of small frames at the end of the sequence. At the decoder, the CPB increases as the channel continues to deliver 5000 bits per second whilst small frames are being decoded and removed from the buffer (Figure 8.13). Eventually, the CPB overflows. Once again, this combination of bitstream and HRD parameters does not conform.

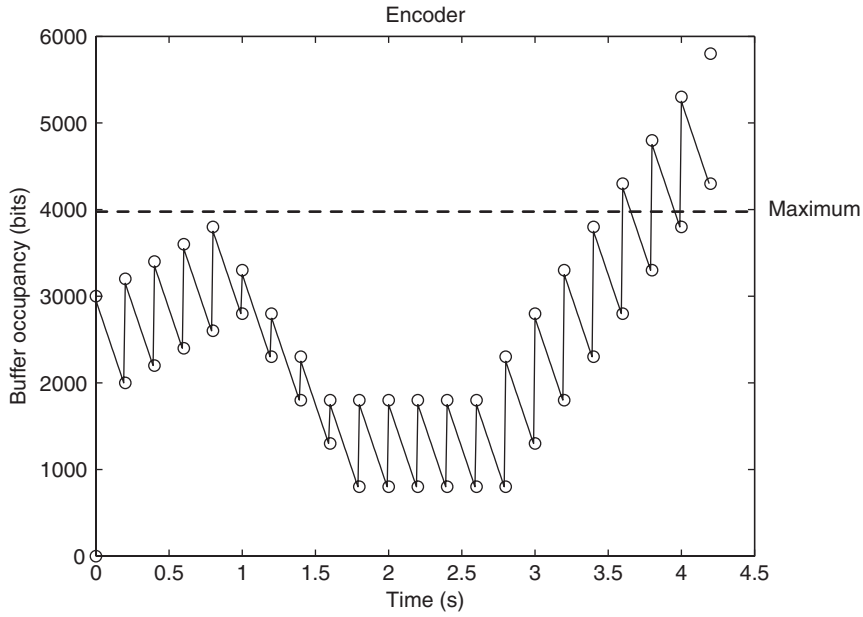


Figure 8.10 HRD Example 2: encoder buffer

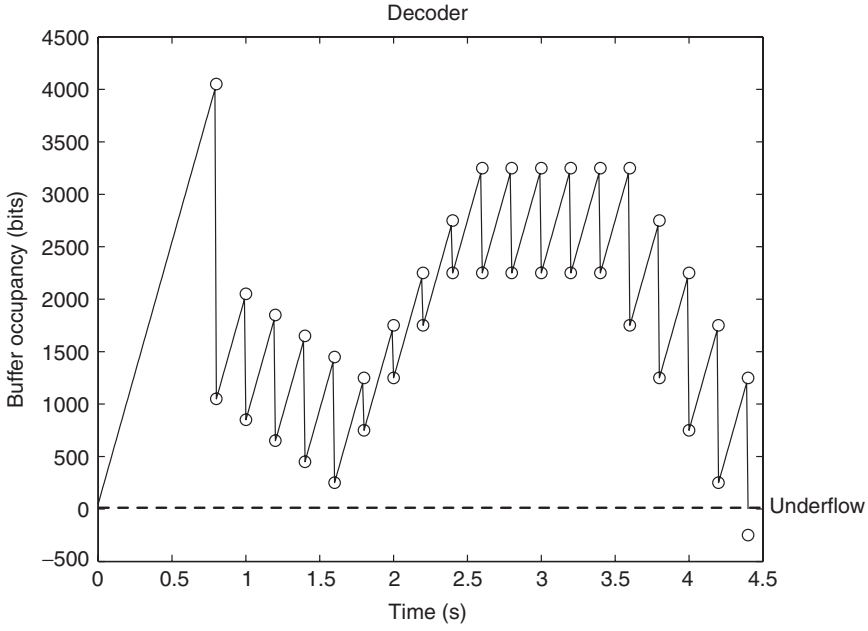


Figure 8.11 HRD Example 2: decoder CPB

Table 8.4 HRD example 3: frame sizes

Frame	Coded size (bits)
. . ..	. . ..
10	1000
11	1000
12	1000
13	1000
14	200
15	200
16	200
17	400
18	400
19	300
20	300
21	300

The HRD as specified in H.264/AVC Annex C is very flexible and can handle situations such as variable bit rate channels, where the delivery rate changes with respect to time, varying CPB removal times corresponding to a decoder that does not operate at a constant frame rate, etc. An H.264 encoder operating at a particular Level is required to produce a bitstream that will not violate the HRD conformance rules.

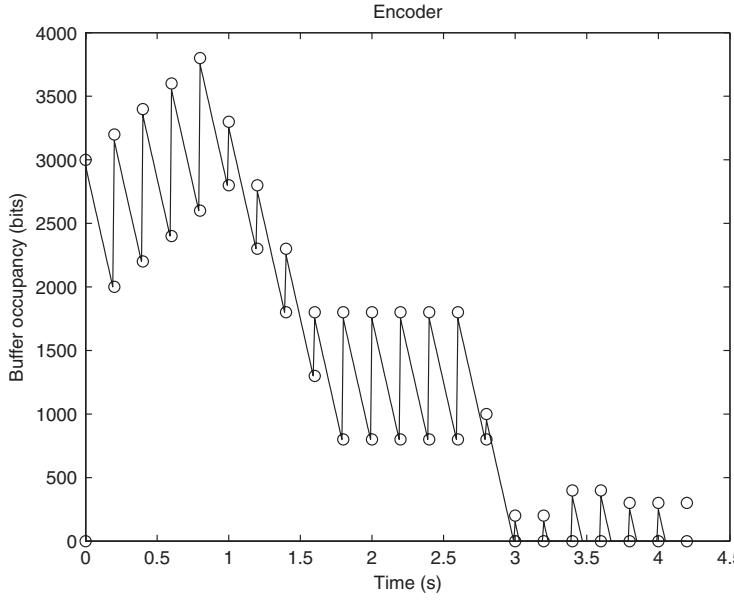


Figure 8.12 HRD Example 3: encoder buffer

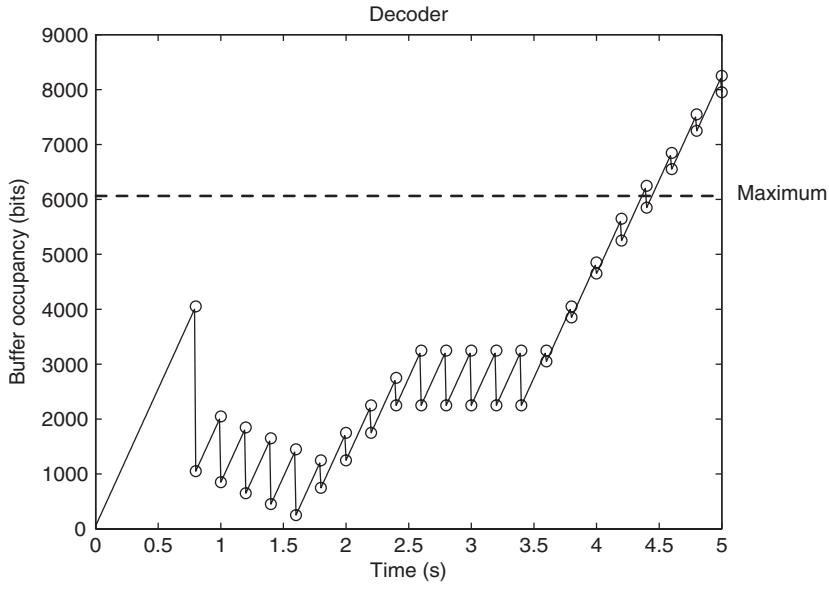


Figure 8.13 HRD Example 3: decoder CPB

8.2.4 Conformance testing

Annex C of H.264/AVC describes how the HRD is used to test conformance. Conformance testing methods are specified in H.264.1 [iv]. This Recommendation describes a series of tests that are intended to verify whether coded bitstreams and video decoders conform to the requirements of H.264/AVC.

8.2.4.1 Testing a bitstream

Bitstream conformance, i.e. whether a coded video bitstream actually conforms to H.264/AVC, may be tested using the Joint Model reference software decoder [v] (Figure 8.14). The bitstream to be tested is decoded by the reference decoder to produce an output video sequence. If there is a problem with the syntax, the HRD behaviour, etc., the reference decoder should indicate this with an error message. Error-free operation implies but does not guarantee conformance.

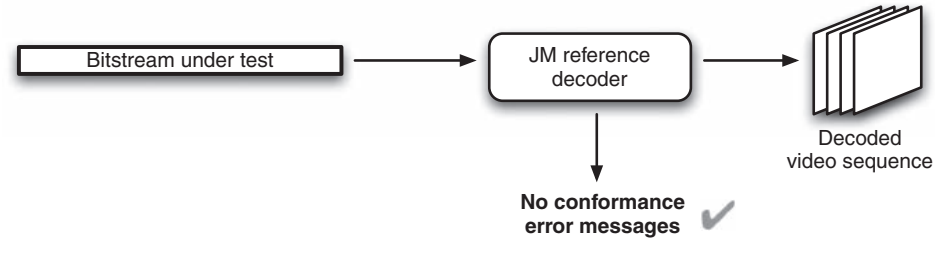


Figure 8.14 Bitstream conformance testing

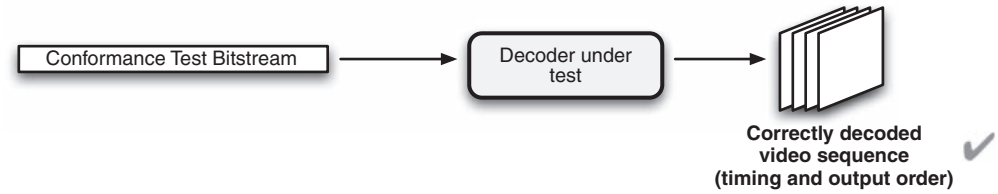


Figure 8.15 Decoder conformance testing

8.2.4.2 Testing a decoder

A decoder should be capable of decoding any bitstream up to its Profile and Level limits (section 8.2.1). Checking that a decoder conforms to H.264/AVC may be carried out by decoding a set of Conformance Test Bitstreams (Figure 8.15). A conforming decoder should produce a correct video sequence, with the frames/fields in the correct order and with the necessary timing relationships between output frames. Conformance bitstreams are available for each of the H.264/AVC Profiles at a range of Levels and may be obtained from the ITU-T [vi] or in draft form from the Joint Video Team archive [vii].

8.2.4.3 Testing an encoder

There is no specific method of testing encoder conformance, since strictly speaking H.264/AVC does not define or specify a video encoder. However, an encoder that is claimed to conform to H.264/AVC should always produce coded bitstreams that meet the conformance requirements of H.264/AVC Annex C and the testing procedures defined in [iv].

8.3 H.264 coding tools for transport support

H.264/AVC is probably best known as a format for efficient video compression. However, in recognition of the fact that most applications of H.264 involve communication or storage of the compressed bitstream, the standard specifies a number of features or tools that are intended to support efficient and robust transport. Redundant slices, Arbitrary Slice Order and Flexible Macroblock Order are supported by the Baseline and Extended Profiles; Data Partitioned Slices, SI Slices and SP Slices are supported by the Extended Profile. It is worth noting that these features have not been widely adopted by commercial H.264/AVC codecs, most of which tend to use the Constrained Baseline, Main and High Profiles.

8.3.1 Redundant slices

A slice marked as ‘redundant’ contains a redundant representation of part or all of a coded frame. In normal operation, the decoder reconstructs the frame from ‘non-redundant’ slices and discards any redundant slices. However, if the primary decoded frame is damaged, e.g. due to a transmission error, the decoder may replace the damaged area with decoded data from a redundant slice if available. Adding redundant slices to a coded bitstream may therefore improve performance in the presence of transmission errors or losses, at the expense of an increase in the number of transmitted bits and hence a loss of compression performance.

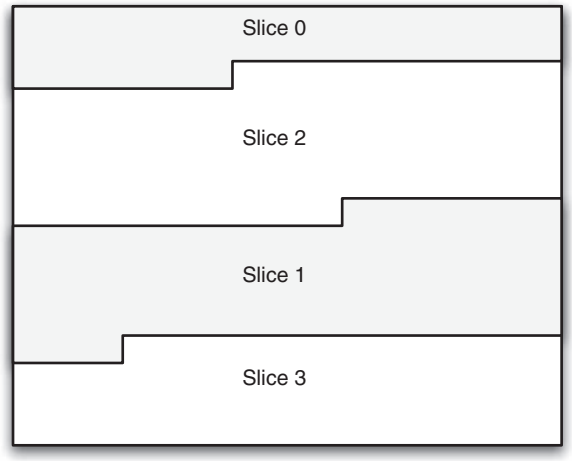


Figure 8.16 Arbitrary Slice Order: Example

8.3.2 Arbitrary Slice Order (ASO)

Arbitrary Slice Order makes it possible for slices in a coded frame to be arranged in any decoding order. ASO is defined to be in use if the first macroblock in any slice in a decoded frame has a smaller macroblock address than the first macroblock in a **previously** decoded slice in the same frame; hence the slices are transmitted in a non-raster order. This may be useful as an aid to decoder error concealment. Figure 8.16 shows an example. Slices are coded in the bitstream in order 0, 1, 2, 3. However, the first macroblock in Slice 2 occurs immediately after the last MB in Slice 0 and has a smaller address, i.e. occurs earlier in the displayed frame, than the first macroblock in Slice 1. For example, if an error or packet loss affects slices 0 and 1, it may be easier for a decoder to conceal the effect of the error using the correctly decoded slices 2 and 3.

8.3.3 Slice Groups / Flexible Macroblock Order (FMO)

Flexible Macroblock Ordering enables macroblocks in a coded frame to be allocated to one of several **slice groups**, each containing a subset of the macroblocks that make up a frame and each containing one or more slices. Within a slice group, MBs are coded in raster order but successive MBs in a slice group are not necessarily adjacent. If a coded frame contains more than one slice group, then Flexible Macroblock Ordering is in use. The allocation of macroblocks is determined by a **macroblock allocation map** that indicates which slice group each MB belongs to. Table 8.5 lists the different types of macroblock allocation maps.

FMO can improve error resilience because each slice can be decoded independently of other slices. For example, if one slice or slice group is lost in a picture using Interleaved ordering, the damaged region may be concealed reasonably effectively using spatial error concealment by interpolating vertically between the decoded macroblocks in the remaining slice group(s).

Table 8.5 Macroblock allocation map types

Type	Name	Description
0	Interleaved	*run_length* MBs are assigned to each slice group in turn (Figure 8.17).
1	Dispersed	MBs in each slice group are dispersed throughout the frame (Figure 8.18).
2	Explicit	A parameter, *slice_group_id*, is sent for each MB to indicate its slice group, i.e. the macroblock map is entirely user-defined.
3	Foreground and background	All but the last slice group are defined as rectangular regions within the frame. The last slice group contains all MBs not contained in any other slice group, the 'background'. In the example in Figure 8.19, group 1 overlaps group 0 and so MBs not already allocated to group 0 are allocated to group 1.
4	Box-out	A rectangular area or box is created starting from the centre of the frame, with the size controlled by encoder parameters and containing group 0; all other MBs are in group 1 (Figure 8.20).
5	Raster scan	Group 0 contains MBs in raster scan order from the top-left and all other MBs are in group 1 (Figure 8.20).
6	Wipe	Group 0 contains MBs in vertical scan order from the top-left and all other MBs are in group 1 (Figure 8.20).

0
1
2
0
1
2
0
1
2

Figure 8.17 FMO: Interleaved map, QCIF, 3 slice groups

0	1	2	3	0	1	2	3	0	1	2
2	3	0	1	2	3	0	1	2	3	0
0	1	2	3	0	1	2	3	0	1	2
2	3	0	1	2	3	0	1	2	3	0
0	1	2	3	0	1	2	3	0	1	2
2	3	0	1	2	3	0	1	2	3	0
0	1	2	3	0	1	2	3	0	1	2
2	3	0	1	2	3	0	1	2	3	0
0	1	2	3	0	1	2	3	0	1	2

Figure 8.18 FMO: Dispersed macroblock map, QCIF, 4 slice groups

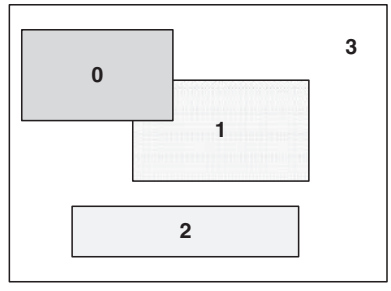


Figure 8.19 FMO: Foreground and Background map, 4 slice groups

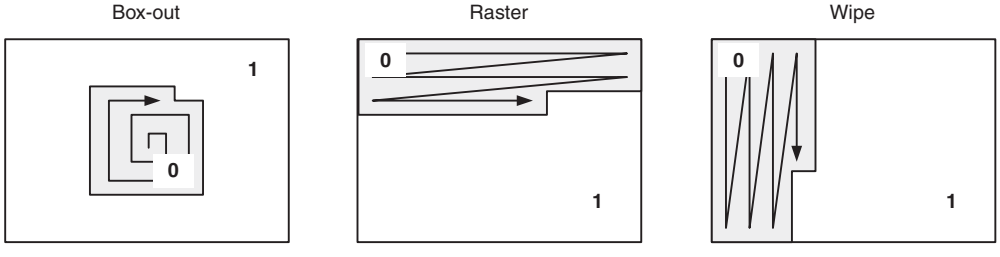


Figure 8.20 FMO: Box-out, Raster and Wipe maps

8.3.4 SP and SI slices

SP and SI slices are specially-coded slices that enable, among other things, efficient switching between video streams and efficient random access for video decoders [viii]. A common requirement is for a video decoder to switch between one of several encoded streams. For example, the same video material is coded at multiple bitrates for transmission across the Internet and a decoder attempts to decode the highest-bitrate stream it can receive but may require to switch automatically to a lower-bitrate stream if the data throughput drops.

Example:

A decoder is decoding Stream A and wants to switch to decoding Stream B (Figure 8.21). For simplicity, assume that each frame is encoded as a single slice and predicted from one reference, the previous decoded frame. After decoding P-slices A_0 and A_1, the decoder wants to switch to Stream B and decode B_2, B_3 and so on. If all the slices in Stream B are coded as P-slices, then the decoder will not have the correct decoded reference frame(s) required to reconstruct B_2, since B_2 is predicted from the decoded frame B_1 which does not exist in stream A. One solution is to code frame B_2 as an I-slice. Because it is coded without prediction from any other frame, it can be decoded independently of preceding frames in stream B and the decoder can therefore switch between stream A and stream B as shown in Figure 8.21. Switching can be accommodated by inserting an I-slice at regular intervals in the coded sequence to create 'switching points'. However, an I-slice is likely to contain much more coded data than a P-slice and the result is an undesirable peak in the coded bitrate at each switching point.

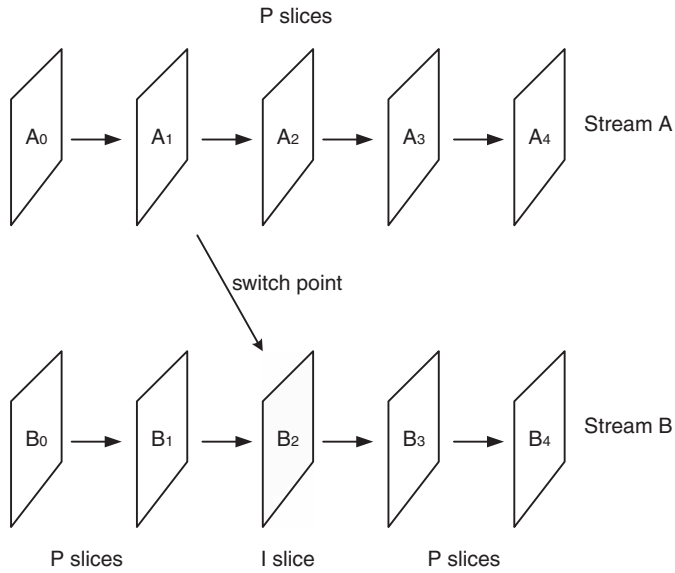


Figure 8.21 Switching streams using I-slices

SP-slices are designed to support switching between similar coded sequences, for example, the same source sequence encoded at various bitrates, without the increased bitrate penalty of I-slices (Figure 8.22). At the switching point, frame 2 in each sequence, there are 3 SP-slices, each coded using motion compensated prediction, making them more efficient than I-slices. SP-slice A_2 can be decoded using reference frame A_1 and SP-slice B_2 can be decoded using reference frame B_2. The key to the switching process is SP-slice AB_2 known as a **switching SP-slice**, created in such a way that it can be decoded using motion-compensated reference frame A_1, to produce decoded frame B_2. This means that the decoder output frame B_2 is identical whether decoding B_1 followed by B_2 or A_1 followed by AB_2. An extra SP-slice is required at each switching point and in fact another SP-slice, BA_2, would be required to switch in the other direction, but this is usually more efficient than encoding frames A_2 and B_2 as I-slices. Table 8.6 lists the steps involved when a decoder switches from stream A to stream B.

Figure 8.23 shows a simplified diagram of the encoding process for SP-slice A_2, produced by subtracting a motion-compensated version of A_1', decoded frame A_1, from frame A_2 and then coding the residual. Unlike a 'normal' P-slice, the subtraction occurs in the transform domain after the block transform. SP-slice B_2 is encoded in the same way (Figure 8.24). A decoder that has previously decoded frame A_1 can decode SP-slice A_2 as shown in Figure 8.25. Note that this is a simplified diagram for clarity; in practice further quantization and rescaling steps are required to avoid mismatch.

SP-slice AB_2 is encoded as shown in Figure 8.26 (simplified). Frame B_2, the frame we are switching to, is transformed and quantized and a motion-compensated prediction is formed from A_1', the frame we are switching from. The 'MC' block in this diagram attempts to find the best match for each MB of frame B_2 **using decoded frame A_1 as a reference**. The motion-compensated prediction is transformed and quantized, then subtracted from the transformed

Table 8.6 Switching from stream A to stream B using SP-slices

Input to decoder	MC reference	Output of decoder
P-slice A_0	[earlier frame]	Decoded frame A_0
P-slice A_1	Decoded frame A_0	Decoded frame A_1
SP-slice AB_2	Decoded frame A_1	Decoded frame B_2
P-slice B_3	Decoded frame B_2	Decoded frame B_3
. . ..	. . ..	. . ..

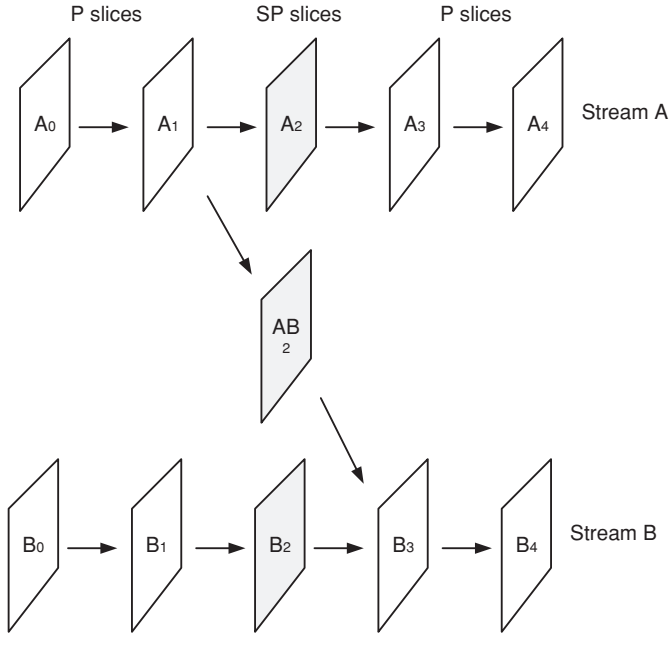


Figure 8.22 Switching streams using SP-slices

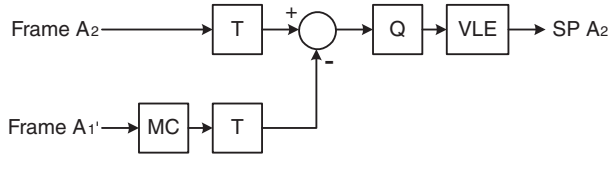


Figure 8.23 Encoding SP-slice A_2 (simplified)

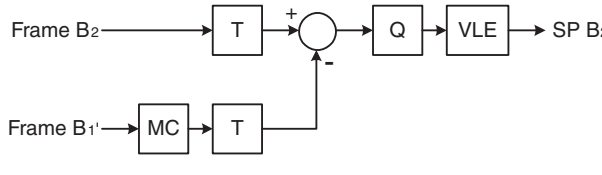


Figure 8.24 Encoding SP-slice B_2 (simplified)

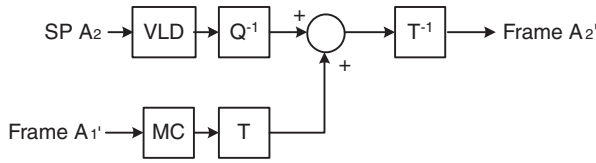


Figure 8.25 Decoding SP-slice A_2 (simplified)

and quantized B_2. Hence in the case of a switching SP slice, subtraction takes place in the quantized transform domain. The residual after subtraction is encoded and transmitted.

A decoder that has previously decoded A_1' can decode SP-slice AB_2 to produce frame B_2' (Figure 8.27). Frame A_1' is motion compensated using the motion vector data encoded as part of AB_2, transformed, quantized and added to the decoded residual, then the result is rescaled and inverse transformed to produce B_2'.

If streams A and B are versions of the same original sequence coded at different bitrates, the motion-compensated prediction of B_2 from A_1', SP-slice AB_2, should be quite efficient. Results show that using SP-slices to switch between different versions of the same sequence is significantly more efficient than inserting I-slices at switching points. Another application of SP-slices is to provide random access and 'VCR-like' functionalities. For example, an SP-slice and a switching SP-slice are placed at the position of frame 10 (Figure 8.28). A decoder can fast-forward from frame A_0 directly to frame A_{10} by first decoding A_0, then decoding switching SP-slice $A_{0\text{-}10}$ which can be decoded to produce frame A_{10} by prediction from A_0.

A further type of switching slice, the SI-slice, is also supported in the Extended Profile. This is used in a similar way to a switching SP-slice, except that the prediction is formed using the 4 × 4 Intra Prediction modes (Chapter 6) from previously-decoded samples of the reconstructed frame. This slice mode may be used, for example, to switch from one sequence to a completely different sequence, in which case it will not be efficient to use motion compensated prediction because there is no correlation between the two sequences.

8.3.5 Data partitioned slices

The coded data that makes up a slice is placed in three separate Data Partitions A, B and C, each containing a subset of the coded slice. Partition A contains the slice header and header data for each macroblock in the slice, Partition B contains coded residual data for Intra and SI slice macroblocks and Partition C contains coded residual data for P, B and SP slice

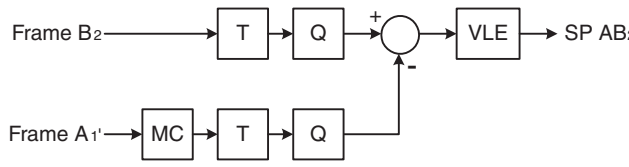


Figure 8.26 Encoding SP-slice AB_2 (simplified)

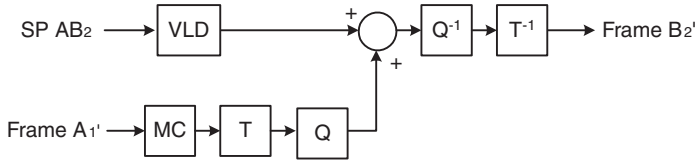


Figure 8.27 Decoding SP-slice AB_2

macroblocks. Each Partition is placed in a separate NAL unit and may therefore be transported separately.

If Partition A data is lost, it is likely to be difficult or impossible to reconstruct the slice, i.e. Partition A is highly intolerant to errors. Partition B is more tolerant to errors, since errors in the intra residual may be concealed at the decoder and Partition C is likely to be the most tolerant to errors, i.e. it contains the least sensitive data, since concealment of errors in inter-coded data is relatively easy. Strategies for improving performance in an error-prone environment include applying unequal error protection to the three partition types, e.g. by applying Forward Error Correction to Partition A and perhaps B, or transporting the partition types over different channels, selecting the most reliable channel for Partition A.

8.4 Transport of H.264 data

8.4.1 Encapsulation in RBSPs, NALUs and packets

H.264 syntax elements are encapsulated into Raw Byte Sequence Payloads (RBSP) and subsequently into Network Abstraction Layer Units (NALU) (Figure 8.29). A sequence of syntax elements such as a coded slice, sequence parameter set or picture parameter set is shown at the top of the figure (Chapter 5). Syntax elements are represented as binary codes with varying lengths (see Chapter 5) and so the sequence of syntax elements may or may not

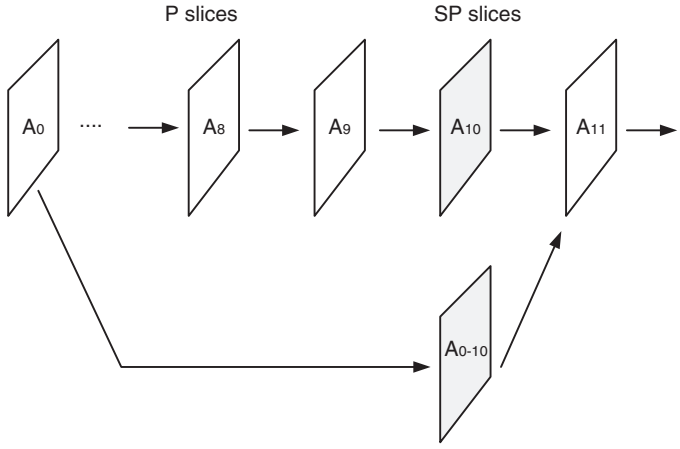


Figure 8.28 Fast-forward using SP-slices

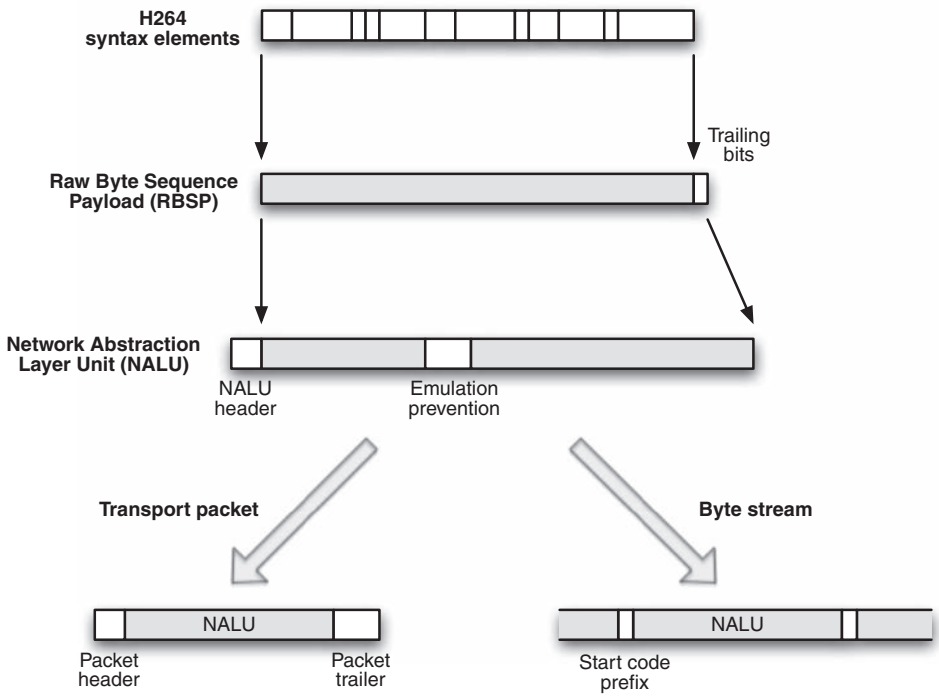


Figure 8.29 Encapsulation of H.264 syntax elements

consist of an integral number of bytes. RBSP trailing bits, a series of zero bits, are added as necessary to create a RBSP that is byte aligned, i.e. contains an integral number of bytes.

A RBSP is encapsulated in a NAL Unit by (i) adding a one-byte NALU header and (ii) inserting Emulation Prevention byte(s) as necessary, in order to prevent a Start Code Prefix occurring in the body of a NAL Unit. An Emulation Prevention byte (00000011 in binary) is inserted by an encoder whenever a 3-byte pattern that is the same as a Start Code Prefix happens to occur within the sequence of syntax elements. The Emulation Prevention byte is detected and removed by a decoder and prevents the decoder from finding an erroneous Start Code in the sequence.

A NALU may be transmitted using a transport protocol in which the NALU forms the payload of a packet or in a byte stream in which NALUs are sent sequentially in a series of bytes (Annex B of the standard). In the case of a byte stream, each NALU is preceded by a Start Code Prefix, a unique 3-byte pattern that cannot occur inside a NALU (see above). A decoder can search for Start Code Prefixes to find the boundaries between NALUs. Extraction of a decodeable H.264 bitstream from a transport packet stream requires the insertion of Start Code Prefixes so that the extracted bitstream can be decoded.

8.4.2 Transport protocols

H.264/AVC does not define a transport mechanism for coded visual data. However, there are a number of possible transport solutions depending on the method of transmission, including the following:

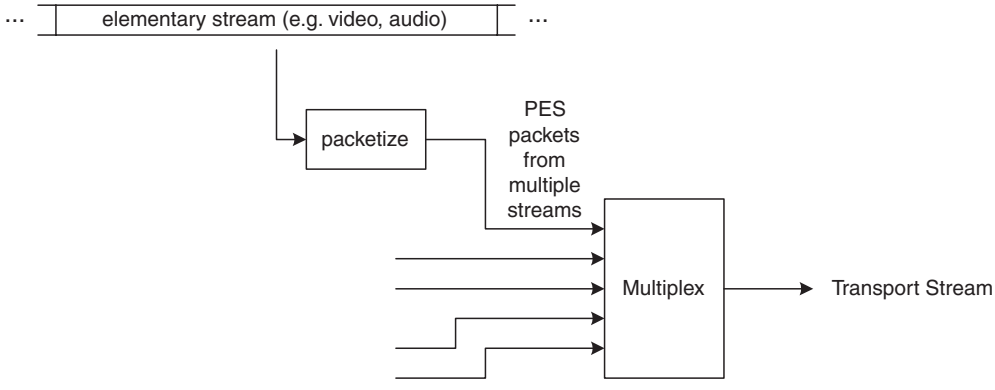


Figure 8.30 MPEG-2 Transport Stream

MPEG-2 Systems: Part 1 of the MPEG-2 standard [ix] defines two methods of multiplexing audio, video and associated information into streams suitable for transmission, Program Streams or Transport Streams. Each data source or **elementary stream** such as a coded video or audio sequence is packetized into Packetized Elementary Stream (PES) packets and PES packets from the different elementary streams are multiplexed together to form a Program Stream, typically carrying a single set of audio/visual data such as a single TV channel, or a Transport Stream which may contain multiple channels (Figure 8.30). The Transport Stream adds both Reed-Solomon and convolutional error control coding and so provides protection from transmission errors. Timing and synchronisation is supported by a system of clock references and time stamps in the sequence of packets. Carriage of an H.264 stream over MPEG-2 Systems is covered by Amendment 3 to MPEG-2 Systems.

Real-Time Protocol: RTP [x] is a packetization protocol that may be used in conjunction with the User Datagram Protocol (UDP) to transport real-time multimedia data across networks that use the Internet Protocol (IP). UDP is preferable to the Transmission Control Protocol (TCP) for real-time applications because it offers low-latency transport across IP networks. However, it has no mechanisms for packet loss recovery or synchronisation. RTP defines a packet structure for real-time data (Figure 8.31) that includes a type identifier to signal the type of CODEC used to generate the data, a sequence number, essential for re-ordering packets that are received out of order, and a time stamp, necessary to determine the correct presentation time for the decoded data. Transporting a coded audio-visual stream via RTP involves packetizing each elementary stream into a series of RTP packets, interleaving these and transmitting them across an IP network using UDP as the basic transport protocol. RTP **payload formats** are defined for various standard video and audio CODECs, including H.264. The NAL structure

	Payload Type	Sequence Number	Timestamp	Unique Identifier	Payload (e.g. Video Packet)

Figure 8.31 RTP packet structure (simplified)

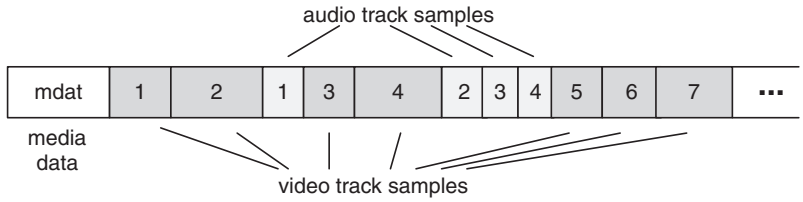


Figure 8.32 ISO Media File

of H.264 (section 8.4.1) has been designed with efficient packetisation in mind, since each NAL unit can be placed in its own RTP packet.

8.4.3 File formats

It is common for single compressed video sequences to be stored in files, simply by mapping the encoded stream to a sequence of bytes in a file. However, storing and playing back combined audio-visual data requires a more sophisticated file structure, especially when, for example, the stored data is to be streamed across a network or when the file is required to store multiple audio-visual objects. The H.264 File Format is designed to store H.264 Video data. It is derived from the ISO Base Media File Format which in turn is based on Apple Computer's QuickTime format.

In the ISO Media File Format, a coded stream such as an H.264 video sequence or an audio stream is stored as a **track**, representing a sequence of coded data items or **samples**, e.g. a coded VOP or coded slice, with time stamps (Figure 8.32). The file formats deal with issues such as synchronisation between tracks, random access indices and carriage of the file on a network transport mechanism.

8.4.4 Coding and transport issues

It has long been recognized that it is necessary to take into account practical transport issues in a video communication system and a number of tools in each standard are specifically designed to address these issues.

Scaling a delivered video stream to support decoders with different capabilities and/or delivery bitrates is addressed by the Scalable Video Coding extension of H.264/AVC (Chapter 10).

Latency is a particular issue for two-way real time applications such as videoconferencing. Tools such as B-slices, coded pictures that use motion-compensated prediction from earlier and later pictures in temporal order, can improve compression efficiency but typically introduce a delay of several frame periods into the coding and decoding 'chain' which may be unacceptable for low-latency two way applications. Latency requirements also have an influence on rate control algorithms since post-encoder and pre-decoder buffers, useful for smoothing out rate variations, lead to increased latency.

Each standard includes a number of features to aid the handling of transmission errors. Bit errors are a characteristic of circuit-switched channels; packet-switched networks tend to suffer from packet losses, since a bit error in a packet typically results in the packet being

dropped during transit. Errors can have a serious impact on decoded quality [xi] because the effect of an error may propagate spatially, distorting an area within the current decoded frame, and temporally, propagating to successive decoded frames that are temporally predicted from the errored frame. Section 8.3 discusses tools that are specifically intended to reduce the damage caused by errors, including data partitioning, designed to limit error propagation by localizing the effect of an error, redundant slices, sending extra copies of coded data, and flexible ordering of macroblocks and slices to make it easier for a decoder to 'conceal' the effect of an error by interpolating from neighbouring error-free data.

8.5 Supplemental Information

Supplemental Enhancement Information (SEI) and Video Usability Information (VUI) are parameters that may be transmitted as part of an H.264 bitstream. SEI and VUI parameters may be useful to a decoder but are not essential to the basic decoding process.

8.5.1 Supplemental Enhancement Information (SEI)

A Supplemental Enhancement Information Message (Table 8.7) is transmitted in an SEI Raw Byte Sequence Payload. Each SEI message is sent as a separate NAL Unit.

8.5.2 Video Usability Information (VUI)

Video Usability Information is transmitted in an optional structure within a Sequence Parameter Set. The VUI parameters syntax structure contains flags and parameters including those summarised in Table 8.8.

8.6 Licensing H.264/AVC

In early 2007, semiconductor giants Qualcomm and Broadcom faced each other in a San Diego courtroom, the culmination of a year-long patent dispute. The questions at issue were whether two Qualcomm patents were essential to the practice of the H.264 video compression standard; whether Broadcom's products had infringed these patents; and whether Qualcomm actually had the right to try and enforce its patents in this context. Why were both firms prepared to commit to millions of dollars of legal costs and to go to the wire in a high-profile court case to fight their respective positions?

Intellectual property, patents and licensing have played an increasingly important part in the recent history of video compression. Video coding is a highly active research and development area and a core technology for a number of billion-dollar industries and so it is perhaps not surprising that there is a high level of patent activity, with a growing number of patents related to video coding. A strong patent on an important video coding concept has the potential to be very lucrative, particularly if the patented technology is adopted in a video coding standard.

Table 8.7 SEI messages

SEI Message	Description
Buffering Period	Controls the operation of the Hypothetical Reference Decoder (HRD) (section 8.2.3).
Picture Timing	Controls the timing of the HRD.
Pan Scan Rectangle	Specifies the location of a rectangle that may be used to implement 'pan-scan' in a display device, i.e. where only part of the decoded picture can be displayed on the device.
Filler Payload	'Dummy' bytes that may be discarded by a decoder. Useful for e.g. avoiding buffer under-run at the decoder.
User Data (Registered)	User data, i.e. data that are not part of the standard, preceded by a code that is registered with the ITU-T.
User Data (Unregistered)	As above, but without any registered code.
Recovery Point	Indicator to help a decoder determine where to start displaying pictures after a transmission error or during random access to a sequence.
Decoded Reference Picture Marking Repetition	Repeats a reference picture marking process signalled earlier (Chapter 5).
Spare Picture	Contains extra, redundant slice group maps (section 8.3.3).
Scene Information	Identifies a sequence of pictures as a video scene. May signal scene transition types such as fade, dissolve, wipe, etc.
Sub-sequence Information	Indicates data dependence hierarchy in terms of a series of layers of coded pictures. Higher-layer pictures are predicted from lower-layer pictures, but not vice versa.
Sub-sequence Characteristics	Indicates characteristics of sub-sequences e.g. target bit rate, frame rate and sub-sequence prediction characteristics.
Full Frame Freeze	Hold or freeze the previously decoded frame.
Full Frame Freeze Release	Cancel the effect of a Full Frame Freeze.
Full Frame Snapshot	Label a frame as a still image 'snapshot' of the video sequence content.
Progressive Refinement Segment Start	Beginning of a set of coded pictures that progressively refine the quality of a single picture, i.e. each decoded residual is an incremental improvement to a single decoded frame.
Progressive Refinement Segment End	Terminates a progressive refinement segment.
Motion Constrained Slice Group	Limit prediction such that there is no motion-compensated prediction from outside a slice group. This may be useful to limit error propagation, for example.
Film Grain Characteristics	Indicates to a decoder a film grain model to apply to the decoded video. Film grain effects or 'graininess' can reduce the performance of a video codec and so it may be useful to apply these after decoding.
Deblocking Filter Display Preference	Indicates whether a decoder should display pictures before or after deblocking filter is applied. The deblocking filter is specified in the standard in order to improve compression performance. It may or may not be preferred to actually show the filtered decoded frames.
Stereo Information	Indicates that the video sequence consists of stereo view pairs of pictures and gives information about the display of these pictures.
Reserved	To be discarded by a decoder, unless specified to have a particular use in future standards.

Table 8.8 Video Usability Information: selected parameters

VUI Parameter(s)	Description
Aspect ratio	Indicates the aspect ratio of the displayed video sequence. This parameter indicates the luma sample aspect ratio which translates into specific display ratios such as 16:9, 4:3, etc.
Overscan	Indicates that the decoded pictures are suitable for overscan, i.e. important content will not be lost if the picture is cropped to fit within a display area.
Video format	Indicates PAL, NTSC, SECAM, MAC or component video formats.
Colour primaries	Indicates the 'chromaticity' of the green, blue, red and white components of the displayed image.
Transfer characteristics	The opto-electronic transfer characteristic of the source picture, characteristics such as display 'Gamma' etc.
Matrix coefficients	Specifies the transform used to derive luma and chroma from red, green and blue colour primaries (Chapter 2).
Chroma sample location	Indicates the location of chroma samples relative to luma samples, if different from the default sampling locations (Chapter 2).
Timing information	Defines the basic units of timing; a 'master clock' for the video decoding processes.
Motion vector range	Specifies maximum motion vector lengths and whether motion vectors may point outside picture boundaries.
Maximum bytes/bits	Places a limit on the number of bytes per coded picture and/or the number of bits per coded macroblock.
HRD parameters	Sets initial parameters for the Hypothetical Reference Decoder (section 8.2.3), other than the default parameter values.

8.6.1 Video coding patents

The popular international standards for video compression such as MPEG-2, MPEG-4 and H.264/AVC make use of the well-known 'hybrid' video encoder model in which a prediction, formed from previously-transmitted frames, is subtracted from the current frame to form a residual which is then transformed using a block transform such as the DCT, quantized and coded for transmission. In parallel, the coded frame is reconstructed and stored for future predictions.

This concept has been in existence for some time. For example, US patent 3679821 [xii], filed April 1970 and issued July 1972, describes an apparatus that 'determines the difference between the momentary value of an incoming frame of video signals and a predicted value of the frame of signals, i.e. the error in prediction, and disperses the difference by transforming it into a spatially homogeneous signal. This transformed signal is then quantized for efficient transmission'. Figure 1 from this patent, redrawn in Figure 8.33, shows an encoder structure that still commonly used in present-day video compression systems. A predicted frame, formed from previously coded data, is subtracted from a video frame S. The difference or residual is transform coded, quantized and transmitted (S_q). The quantized output is inverse transformed and added to the prediction to form a 'Reconstituted Signal' which is used to form further predicted frame(s).

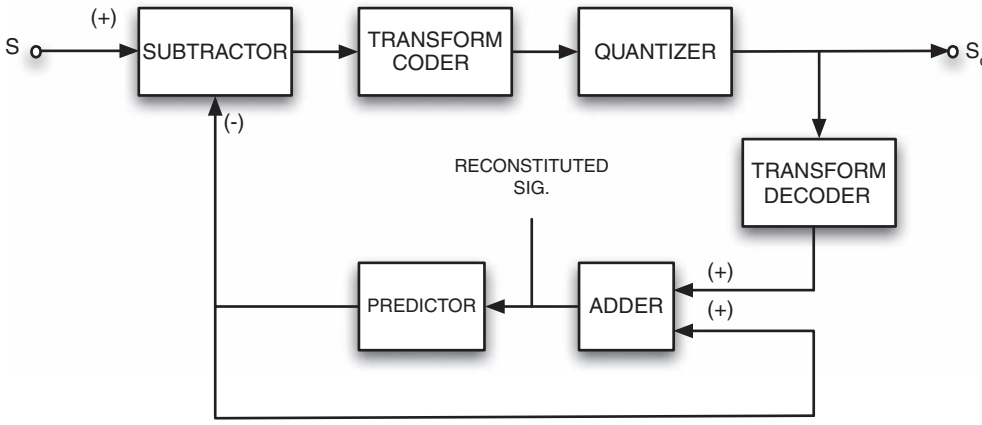


Figure 8.33 Block diagram from US patent 3679821 (redrawn)

By the 1990s, improvements in processing and storage capabilities of integrated circuits and the development of international standards for image and video compression led to a growing market for video compression products and an increased level of research and development activity. This is reflected by a sharp rise in the number of published patents from the early 1990s when the H.261 and MPEG-1 standards were published, peaking around 1998 by which time MPEG-2 products were becoming firmly established in the market and continuing at a steady level thereafter (Figure 8.34). To date (early 2010), the US Patent Database records over 7000 patents featuring the terms 'video compression' or 'video coding'.

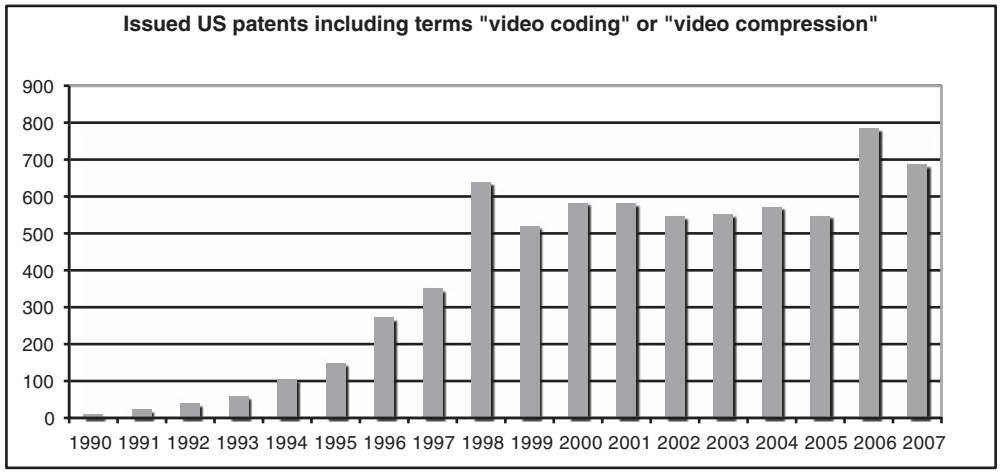


Figure 8.34 Issued US patents including the terms 'video coding' or 'video compression', 1990–2007. Source: USPTO patent database.

8.6.2 Video coding standards and patents

Given the number of published patents in the field of video compression, it is likely that an implementation of a video compression system, particularly one that uses the popular hybrid model of motion compensated prediction, transform and entropy coding, may fall into the scope of one or more patents. This is a concern for manufacturers of products that incorporate video coding technology, since the grant of a US patent confers 'the right to exclude others from making, using, offering for sale, or selling the invention throughout the United States or importing the invention into the United States' (source: USPTO, www.uspto.gov) and similar rights are conferred by patents granted in other jurisdictions. It may therefore be necessary to license the right to use the technology covered by the patent. A complex video coding standard such as H.264/AVC may fall within the scope of a large number of patents held by different parties, raising the prospect of a 'patent thicket' in which it is difficult and costly to negotiate licenses to all the necessary patents.

This problem has been addressed in recent video coding standards in a number of ways. First, the standard-setting bodies such as ITU-T attempt to avoid including IPR (Intellectual Property Rights) in a published standard unless the IPR is licensable on RAND (Reasonable And Non Discriminatory) terms. Second, the group responsible for preparing a draft standard for publication, e.g. the MPEG, VCEG or JVT committees, requests a party proposing technical elements of the draft standard to disclose whether it holds IPR relating to the proposal and whether it is prepared to license this on RAND terms. Third, any party holding IPR believed to be necessary to a published standard is encouraged to declare to the standards-setting body whether it is prepared to license this IPR on RAND terms. The aim is to clarify the IPR position prior to publication of a new standard as far as is possible.

A trend in recent years has been the emergence of 'pool' licenses for IPR relating to a published video coding standard. A third party representing the interests of a number of patent holders sets out terms for licensing the right to use the patented technology in implementations of a published standard. This is intended to provide a 'one stop shop', a single license agreement that covers a large number of patents claimed to be essential to practicing the standard.

8.6.3 Licensing H.264/AVC patents

An H.264 decoder has to implement decoding algorithms specified in the H.264/AVC standard. Whilst the standard does not define encoder operation, in practice an H.264 encoder is likely to implement specific coding algorithms to meet the requirements of the standard. Certain encoding and decoding algorithms implemented in H.264 codecs may fall within the scope of some of the many published video coding patents.

Initially there were two separate patent pools in operation, one managed by Via Licensing and the other by MPEG-LA, each representing some but not all of the organizations claiming to hold patents essential to H.264. Via Licensing later withdrew its 'pool' license leaving MPEG-LA to offer a license to its pool of patents. MPEG-LA's website states that its goal is 'to provide worldwide access to as much AVC essential intellectual property as possible', i.e. it does **not** claim to include all essential IPR in its license. Any party may submit patents for evaluation and possible inclusion in the MPEG-LA license pool.

MPEG-LA provides a license to several hundred patents owned by over 20 organizations, claimed to be essential to H.264/AVC implementations.

Under MPEG-LA's published terms, license fees are required to be paid by manufacturers of encoders and decoders and by suppliers of H.264 coded content, e.g. discs and other media, pay-per-view and subscription services, broadcasting services. No distinction is made between implementations that use different subsets of H.264 functionality, e.g. decoding only; Profiles that are subsets of the available tools; etc.

8.7 Summary

Many of the features of H.264/AVC go beyond basic video compression, covering aspects such as encoder/decoder interoperability, decoder computational capacity, robust transport and storage and display parameters. Interoperability is supported through the use of Profiles, defined sub-sets of coding tools that may be useful for particular classes of application, and Levels, limits on the computational and storage requirements of a decoder and is checked by conformance tests using a Hypothetical Reference Decoder. Reliable transport and storage is inherent in the packet-based structure of an H.264 bitstream and the signalling of key parameters in separate Parameter Sets. Optional coding tools make it possible to further increase reliability of transmission, though many of these tools have not been widely adopted in commercial H.264 codecs. The definition of methods for transporting and storing H.264 content using common transport protocols and file formats has helped the rapid and widespread adoption of the standard. Similarly, the early publication of license terms by key patent holders has helped ease the adoption of H.264/AVC in commercial, large scale consumer applications.

8.8 References

i. Recommendation ITU-T H.264 | ISO/IEC 14496-10:2009, ' Advanced Video Coding for generic audio-visual services', March 2009.
ii. G. Sullivan and A. Tourapis, ' Constrained Baseline Profile and Supplemental Enhancement Information', Draft Amendment to H.264, ISO/IEC JTC1/SC29/WG11 Document N10152, October 2008.
iii. J. Ribas-Corbera, P. A. Chou and S.L. Regunathan, 'A generalized hypothetical reference decoder for H.264/AVC', *IEEE Trans. Circuits and Systems for Video Technology*, vol. 13, no, 7, July 2003, Page(s): 674–687
iv. ITU-T Recommendation H.264.1: Conformance specification for H.264 advanced video coding (June 2008).
v. Joint Model reference software version 16.0, http://iphome.hhi.de/suehring/tml/, July 2009.
vi. International Telecommunication Union, http://www.itu.int/
vii. Draft conformance bitstream files, http://wftp3.itu.int/av-arch/jvt-site/draft_conformance/
viii. M. Karczewicz and R. Kurceren, 'A Proposal for SP-Frames', ITU-T SG16/6 document VCEG-L27, Eibsee, Germany, January 2001.
ix. ISO/IEC 13818, 'Information technology: generic coding of moving pictures and associated audio information', 1995 (MPEG-2).
x. IETF RFC 1889, 'RTP: A transport protocol for real-time applications', January 1996.
xi. A. H. Sadka, *Compressed Video Communications*. John Wiley & Sons Ltd, 2002.
xii. US Patent 3679821, 'Transform Coding of Image Difference Signals', 1972.

9

H.264 performance

9.1 Introduction

The driving force behind the widespread adoption of H.264/AVC is its potential for significantly better compression performance than older formats such as MPEG-2 and MPEG-4 Visual. However, getting the best possible performance from an H.264 codec is not a straightforward task. First, despite the fact that H.264/AVC is an industry standard, there is a wide variation between practical implementations of H.264 codecs, with a corresponding variation in coding performance. Second, the large number of coding options supported by H.264 introduces a problematic trade-off between compression and computation. Achieving the best possible compression performance from an H.264 codec can result in a computational cost that is prohibitive for practical applications.

Probably the best way to understand the performance trade-offs and the capabilities of H.264/AVC is to experiment. Fortunately, a number of public-domain implementations are available that make this possible. The Joint Video Team, responsible for developing and maintaining the standard, publish a reference software implementation of H.264/AVC, the Joint Model (JM) codec. The JM is intended to be a complete and accurate implementation of all the features of H.264. It is not suitable for a practical, real-time coding application but is a useful reference tool to test the potential of H.264 and to check inter-operability between codecs and bitstreams. Using the JM and/or other H.264 implementations, it is possible to code and decode video sequences and to test the effect of the various tools and features of the standard.

Most applications of H.264 place constraints on the bitrate of the coded video sequence. For example, broadcast channels have a fixed capacity in terms of bits per second; internet streaming can handle a variable bitrate, but only within certain upper and lower limits; conversational applications such as video calling require minimal end-to-end delays; and so on. For these reasons, practical implementations of H.264 generally require a rate control algorithm to constrain the coded bitrate within certain limits.

Getting the best performance from an H.264/AVC codec generally involves selecting the best coding options or coding mode for each unit of data in the video bitstream. This process of mode selection is fundamental to achieving good compression performance. Because of

The H.264 Advanced Video Compression Standard, Second Edition Iain E. Richardson
© 2010 John Wiley & Sons, Ltd

this, there has been a significant amount of research in finding good compromises between (a) effective mode selection, and hence good rate-distortion performance and (b) realistic levels of computational complexity.

9.2 Experimenting with H.264

9.2.1 The JM Reference Software

9.2.1.1 Overview

The Joint Video Team (JVT), the group responsible for developing and maintaining H.264/AVC, publishes a reference implementation of the standard as a C software program, known as the **Joint Model** (JM). At the time of writing, the latest version (16.0) can be downloaded from http://iphome.hhi.de/suehring/tml/ [i]. An older version of the software is published as ITU-T standard H.264.2 [ii]. The JM software manual describes the operation and parameters of the software in detail [iii] and a detailed description of many of the coding algorithms incorporated into the JM software can be found in a JVT document published in 2005 [iv].

The JM software consists of an encoder (*lencod*) that encodes a source video file into a coded H.264 file and a decoder (*ldecod*) that decodes an H.264 file into a decoded video file. The encoder and decoder are each controlled by a parameter file with default names *encoder.cfg* and *decoder.cfg*. The encoder creates a reconstructed video file, a copy of the decoded video file, and can optionally generate a trace file that records every syntax element of the coded sequence (Figure 9.1).

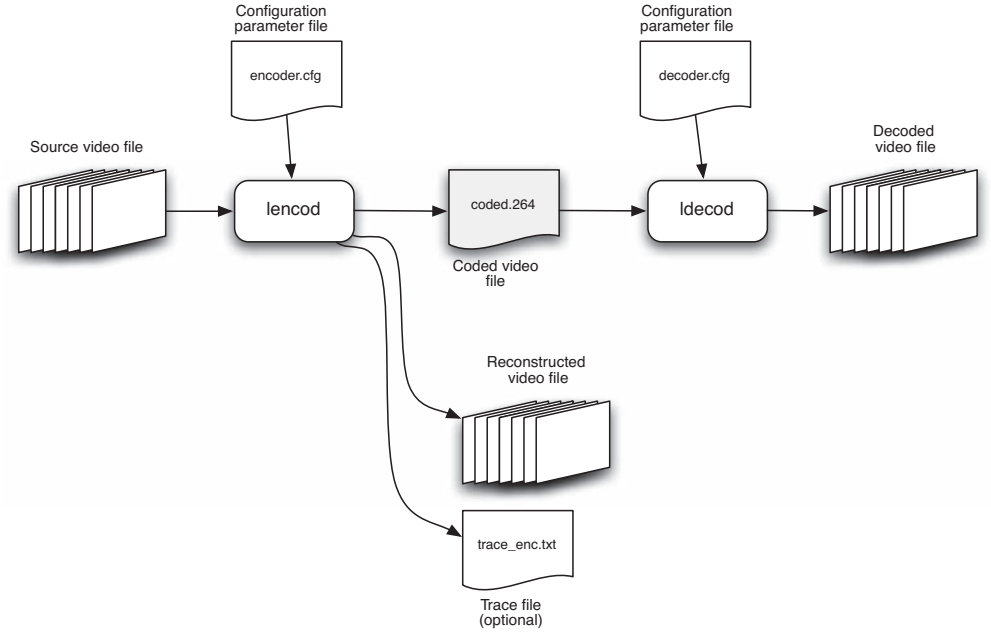


Figure 9.1 JM software operation

Table 9.1 JM software directory structure

Directory	Description
bin or build	Executable files lencod, ldecod
doc	Documentation
lcommon	Source (C) and object code files common to encoder and decoder
ldecod	Decoder source and object code files
lencod	Encoder source and object code files

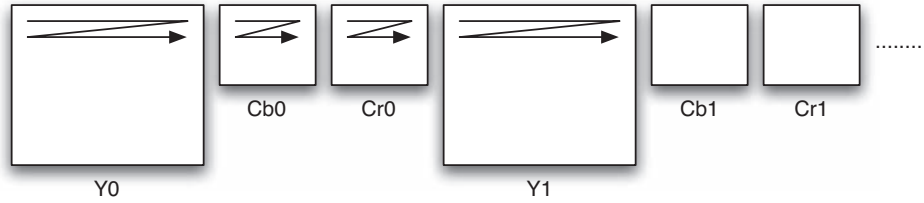


Figure 9.2 Planar YCbCr file format, 4:2:0 sampling

9.2.1.2 File formats

Source, reconstructed and decoded video files are in 'raw' YCbCr format, in which the luma, Cb and Cr samples are stored in the video file with no header or other information. Various sample orders are supported but the default is planar order, with each component of a frame stored in raster scan order, starting with frame 0 (Figure 9.2).

9.2.1.3 Basic operation

The JM software 'unpacks' and compiles into the directory structure shown in Table 9.1.

Example

Use the JM software to code 60 frames of a QCIF video sequence.

1. Download and unpack the JM software. Follow the instructions in *readme.txt* to compile lencod and ldecod.
2. Copy a QCIF source file into the *bin* directory. Various test video files are widely available: for example, *container.qcif*.
3. Create a configuration file. Start with one of the example files in the *bin* directory, e.g. encoder_baseline.cfg. Copy it and give it a new name, e.g. encoder_1.cfg.
4. Edit the new configuration file encoder_1.cfg (Figure 9.3). Changes are shaded; change the input and output file names, the number of frames to be encoded and set the quantizer parameter to 32.
5. Open a command prompt in the *bin* directory.
6. Run the encoder by typing:

```
lencod –d encoder_1.cfg
```

```
################################################
# Files
################################################
InputFile              = "container.qcif "       # Input sequence
InputHeaderLength      = 0      # If the inputfile has a header, state…
StartFrame             = 0      # Start frame for encoding. (0-N)
FramesToBeEncoded      = 60     # Number of frames to be coded
FrameRate              = 30.0   # Frame Rate per second (0.1-100.0)
SourceWidth            = 176    # Source frame width
SourceHeight           = 144    # Source frame height
SourceResize           = 0      # Resize source size for output
OutputWidth            = 176    # Output frame width
OutputHeight           = 144    # Output frame height
TraceFile              = "trace_enc.txt"      # Trace file
ReconFile              = "container_rec.qcif"
OutputFile             = "container.264"
StatsFile              = "stats.dat"          # Coding statistics file
################################################
# Encoder Control
################################################
ProfileIDC             = 66  # Profile IDC (66=baseline, 77=main, 88=extended…
IntraProfile           = 0   # Activate Intra Profile for FRExt (0: false, 1: true)
LevelIDC               = 40  # Level IDC   (e.g. 20 = level 2.0)
IntraPeriod            = 0   # Period of I-pictures   (0=only first)
IDRPeriod              = 0   # Period of IDR pictures (0=only first)
AdaptiveIntraPeriod    = 1   # Adaptive intra period
AdaptiveIDRPeriod      = 0   # Adaptive IDR period
IntraDelay             = 0   # Intra (IDR) picture delay (i.e. coding structure of PPIPPP... )
EnableIDRGOP           = 0   # Support for IDR closed GOPs (0: disabled, 1: enabled)
EnableOpenGOP          = 0   # Support for open GOPs (0: disabled, 1: enabled)
QPISlice               = 32  # Quant. param for I Slices (0-51)
QPPSlice               = 32  # Quant. param for P Slices (0-51)
FrameSkip              = 0   # Number of frames to be skipped in input
…
…
```

Figure 9.3 JM encoder configuration file

```
------------------------------ JM 16.0 (FRExt) ------------------------------
 Input YUV file                    : container.qcif
 Output H.264 bitstream            : container1.264
 Output YUV file                   : container_rec.qcif
 YUV Format                        : YUV 4:2:0
 Frames to be encoded I-P/B        : 60/0
 Freq. for encoded bitstream       : 30.00
…
…
-------------------------------------------------------------------------------
Frame     Bit/pic    QP   SnrY    SnrU    SnrV    Time(ms) MET(ms) Frm/Fld Ref
-------------------------------------------------------------------------------
00000(NVB)     160
00000(IDR)   17896   32  34.357  39.507  39.009        26       0    FRM    1
00001( P )     304   32  34.123  39.594  39.083        46      21    FRM    1
00002( P )     512   32  34.218  39.565  39.116        63      36    FRM    1
00003( P )     240   32  34.023  39.566  39.108        52      25    FRM    1
00004( P )     480   32  34.052  39.632  39.120        53      27    FRM    1
…
…
------------------ Average data all frames  ----------------------------------

 Total encoding time for the seq.  :   2.953 sec (20.32 fps)
 Total ME time for sequence        :   1.383 sec

 Y { PSNR (dB), cSNR (dB), MSE }   : {  33.578,  33.573,  28.56274 }
 U { PSNR (dB), cSNR (dB), MSE }   : {  39.404,  39.402,   7.46286 }
 V { PSNR (dB), cSNR (dB), MSE }   : {  38.910,  38.908,   8.36233 }

 Total bits                        : 50712 (I 17896, P 32656, NVB 160)
 Bit rate (kbit/s)  @ 30.00 Hz     : 25.36
 Bits to avoid Startcode Emulation : 0
 Bits for parameter sets           : 160
 Bits for filler data              : 0

-------------------------------------------------------------------------------
Exit JM 16 (FRExt) encoder ver 16.0
```

Figure 9.4 JM encoder output display

This will produce an output listing similar to that shown in Figure 9.4. Only selected lines are shown.

For each coded frame, the frame type, IDR/I or P in this case, number of coded bits, quantization parameter (QP), PSNR or 'Snr' for the components Y, U and V or Y, Cb, Cr and coding time are listed. Average PSNR and bitrate are listed at the end of the coded sequence. This is a relatively low-bitrate coded sequence at 21.85 kbps.

7. Run the decoder by typing:

```
ldecod –i container1.264 –o container_dec.qcif –r container.qcif
```

Input is container1.264; output is container_dec.qcif; use the original container.qcif as a reference for PSNR calculations.

The visual quality of the decoded QCIF file can be examined using a YUV viewer, a program that plays back a YUV file. A number of YUV viewers are available for download. Note that the Reconstructed file, container_rec.qcif in this case, and the Decoded file container_dec.qcif are identical. Figure 9.5 shows frame 56 from each QCIF file. Note that:

(i) The decoded/reconstructed frames have less detail due to quantization during encoding. A lower QP would improve decoded quality at the expense of a higher bitrate.
(ii) The decoded and reconstructed frames are identical.

Hence it is **not** actually necessary to run the decoder (ldecod) to view the decoded quality, since the encoder generates a decoded video file as well as all the necessary statistics such as bitrate and PSNR required to analyze coding performance.

9.2.1.4 Advanced operation

The large number of optional parameters in *encoder.cfg* give the user detailed control over the operation of the JM encoder. The main sections and their effect on encoder operation are as follows (Table 9.2). Note that some sections e.g. B slice parameters are not present in the Baseline configuration file and that the JM software continues to be developed and so these sections and their content may change.

Figure 9.5 Original, reconstructed and decoded frames, container.qcif, QP=32

Table 9.2 Selected parameter sections in the JM encoder configuration file

Section	Description
Files	Input and output file names; frame size (Y component); number of frames to be encoded; source frame rate, necessary to correctly set rate control parameters.
Encoder control	Basic control parameters. Profile and Level (Chapter 8); I-slice control; I and P slice quantization parameters (QP); motion estimation control: search range, reference frames, partition sizes.
B slices	Frequency of B slices; quantization parameter; Direct Mode and reference control; hierarchical B slices ('PyramidCoding'); bi-predictive motion estimation. See Chapter 6.
Output control	CAVLC ('UVLC') or CABAC entropy coding (Chapter 7); output file mode.
CABAC context initialization	Controls CABAC context models (Chapter 7).
Interlace handling	Coding of interlaced fields: Picture and Macroblock Adaptive Frame/Field Coding (Chapter 5).
Weighted prediction	Controls weighted prediction (Chapter 6).
Picture based multi-pass encoding	Enables multiple coding of each picture e.g. using different QPs, encoder chooses optimum 'pass'.
Deblocking filter parameters	'DFParametersFlag' determines whether any of the following parameters are sent, to control the operation of the deblocking or loop filter (Chapter 6). The default is for the loop filter to be enabled.
Error resilience/slices	SliceMode controls the distribution of slices in each frame; other parameters control Slice Groups and Redundant Slices (Chapter 8).
Search Range/RD Optimization	The most useful parameter here is RDOptimization which controls the use of rate-distortion-optimized (RDOpt) mode selection.
Explicit Lambda Usage	Modify Lambda (λ) parameter, used in Rate Distortion Optiimized mode selection.
Additional Stuff	These parameters are not likely to be of general use.
Rate Control	Enable of disable rate control. If enabled, the encoder attempts to maintain a constant bitrate through automatically varying the quantization parameter (QP).
Fast Mode Decision	Optional algorithms for speeding up coding mode decisions.
Rounding Offset control	Implements adaptive rounding during encoder quantization, i.e. adaptive adjustment of the quantizer thresholds. See [v].
Fast Motion Estimation parameters	Control parameters for the various motion estimation search modes supported by the JM encoder.
SEI Parameters	Generate SEI messages. See Chapter 8.
VUI Parameters	Insert VUI parameters. See Chapter 8.

Parameter examples:

1. Fast encoding:
   To speed up the encoding of a sequence, limit the number of reference frames (NumberReferenceFrames) and the motion search area (SearchRange); use fast motion estimation (e.g. SearchMode = 1); disable rate-distortion optimized mode selection (RDOptimization = 0) or use 'fast' mode (RDOptimization = 2).
2. High quality encoding:
   To maximize the quality of a coded sequence, copy and edit a Main Profile configuration file; use B slices; use a large number of reference frames, a large SearchRange and enable all partition sizes (InterSearchNxN); disable fast motion estimation (SearchMode = 0 or −1); enable CABAC (SymbolMode = 1); use High Complexity rate-distortion optimized mode selection (RDOptimization = 1). Note that Main Profile must be selected if CABAC is enabled or if B slices are used (ProfileIDC = 77).
3. B slices:
   To insert two B slices between successive I or P slices, creating an IBBPBBP... prediction structure, see example in Chapter 6, copy and edit a Main Profile configuration file (ProfileIDC = 77); set NumberBFrames = 2 and set the QP for B slices (QPBSlice).

9.2.1.5 Trace file

It is possible to generate a 'trace' file during encoding, default name trace_enc.txt, by changing the following line in header file *defines.h* :

```
#define TRACE 1
```

prior to compiling the encoder.

Every syntax element produced by the encoder is listed in trace_enc.txt. This is a useful analysis tool but note that the trace_enc.txt file tends to be very large and will considerably slow down the speed of encoding. A sample trace file section is shown in Table 9.3 and more examples can be found in Chapter 5. Each line indicates the current bit count, the current NAL Unit type, SPS in this case, the parameter to be coded, the binary code of the parameter and the numerical value. For example, *num_ref_frames* has the value 5 and is coded as the Exp-Golomb codeword 00110 (Chapter 7).

9.2.2 Other software encoders/decoders

H.264/AVC encoders and decoders are available for a range of platforms including Windows/Linux/Mac, DSP and embedded, ASIC cores and hardware ICs. The JM reference codec runs very slowly on most platforms and is intended for conformance testing and research rather than as a practical real-time codec. The public-domain ×264 codec[1] is used in a number of practical coding applications and performs well in terms of bitrate, picture quality and speed of processing [vi]. For example, Figure 9.6 shows a section of a coded frame from two sequences. The left-hand version was coded using the JM reference encoder, Baseline profile, CIF source,

[1] Note that whilst ×264 may be freely downloaded, commercial usage of any H.264 codec may be subject to license fee claims, see Chapter 8.

Table 9.3 Section of trace file showing Sequence Parameter Set

@0	SPS:	profile_idc	01001101	(77)
@8	SPS:	constrained_set0_flag	0	(0)
@9	SPS:	constrained_set1_flag	0	(0)
@10	SPS:	constrained_set2_flag	0	(0)
@11	SPS:	constrained_set3_flag	0	(0)
@12	SPS:	reserved_zero_4bits	0000	(0)
@16	SPS:	level_idc	00011110	(30)
@24	SPS:	seq_parameter_set_id	1	(0)
@25	SPS:	log2_max_frame_num_minus4	1	(0)
@26	SPS:	pic_order_cnt_type	1	(0)
@27	SPS:	log2_max_pic_order_cnt_lsb_minus4	00101	(4)
@32	SPS:	num_ref_frames	00110	(5)
@37	SPS:	gaps_in_frame_num_value_allowed_flag	0	(0)
@38	SPS:	pic_width_in_mbs_minus1	000010110	(21)
@47	SPS:	pic_height_in_map_units_minus1	000010010	(17)
@56	SPS:	frame_mbs_only_flag	1	(1)
@57	SPS:	direct_8x8_inference_flag	1	(1)
@58	SPS:	frame_cropping_flag	0	(0)
@59	SPS:	vui_parameters_present_flag	0	(0)

QP = 26. The right-hand version was coded using ×264 with the same settings. The file sizes and image quality are almost identical; in fact ×264 produces a slightly smaller file. For this sequence, the JM encoder runs at 5 frames per second, much slower than real-time, and the ×264 encoder runs at 42 frames per second, i.e. faster than real-time, on the same computer. In this example, ×264 gives good compression performance at a much faster encoding speed than the JM.

Independent evaluation tests are a good source of comparisons between video coding solutions. *Doom9*'s 2005 codec comparison was 'won' by ×264, closely followed by Ateme's H.264 codec [vii]. Moscow State University has carried out a series of evaluation competitions

Figure 9.6 Section of coded frame, JM encoder (left), ×264 encoder (right)

for H.264 and MPEG-4 video codecs, the most recent of which concludes that ×264 and Mainconcept's H.264 codec[2] were the best of the codecs tested [viii].

9.2.3 H.264 stream analysis

A syntax or stream analyzer reads an H.264/AVC bitstream and extracts and displays information about the coding choices, etc. Using the JM encoder in TRACE mode (section 9.2.1) extracts this type of information at a very low level but the amount of information generated makes it difficult to interpret. Commercially available stream analysers can provide useful information in a graphical form and can extract higher-level performance indicators. At the time of writing, companies offering H.264/AVC analysis tools include Elecard, Mindego, Sencore and Thomson Grass Valley.

Example 1: Baseline Profile

Figure 9.7 shows a screenshot of Elecard's StreamEye analyzer software. The main (top) window shows a profile of the bitstream, in this case, a Baseline Profile sequence that starts with an I/IDR slice followed by P slices. A single frame is displayed in the lower-left window with overlays such as partition sizes, motion vectors and macroblock types. In this example, a Macroblock Info window (lower-right) displays coding information about a selected macroblock.

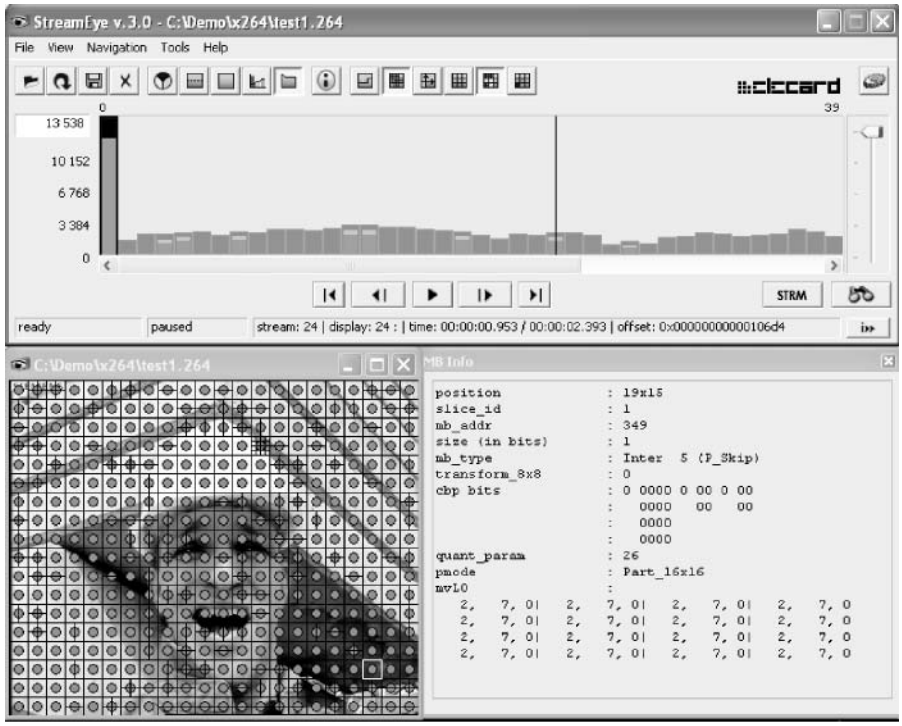


Figure 9.7 Screenshot of an H.264 stream analyzer: Baseline Profile frame. Reproduced by permission of Elecard

[2] Mainconcept is now owned by DivX.

Note that the first frame of the sequence, the I slice, contains a large number of coded bits; subsequent P slices are coded with a much lower number of bits. This is because motion compensated inter prediction is much more efficient than Intra prediction. Note also that the number of bits in each P slice varies. In general, a frame containing more motion and/or detail will require more bits than a frame containing less motion or detail.

The selected macroblock near the lower-right of the frame is coded in P_Skip mode, i.e. no transform coefficients or motion vector differences are sent and the macroblock is reconstructed using motion compensated prediction with a motion vector (mvL0) predicted from previously-sent vectors (Chapter 6).

Example 2: Main Profile

The same sequence, CIF, 30 frames per second, is coded using Main Profile tools (Figure 9.8). The first frame is coded as an I slice and subsequent frames are coded as P or B slices.

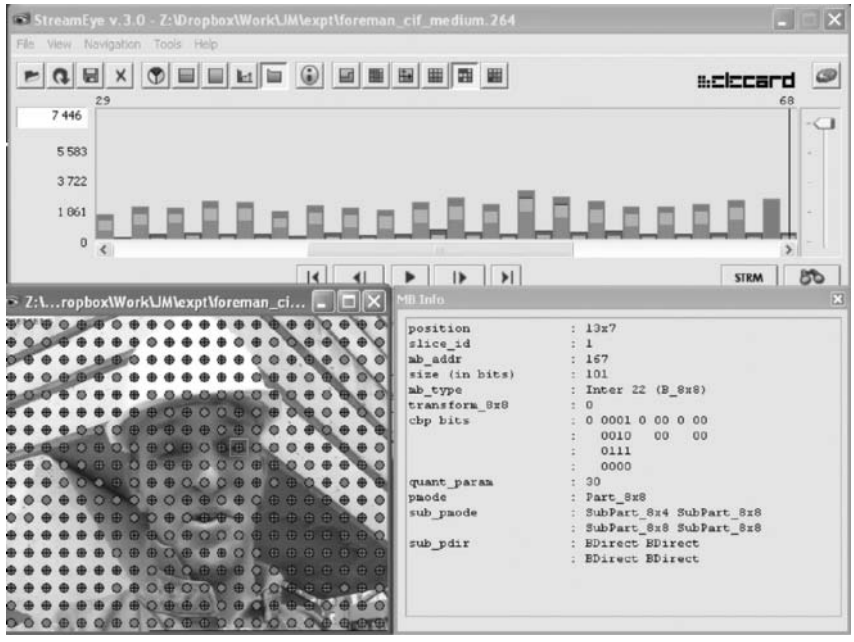


Figure 9.8 Screenshot of stream analyzer: Main Profile frame. Courtesy of Elecard

The top window shows the number of bits per frame; the first frame is not shown on this graph. The coded P slices, every second frame, are significantly larger than the B slices, demonstrating that bi-predicted inter prediction is more efficient than prediction from a single reference. In fact, most of the macroblocks in the B slice shown in the figure are skipped, i.e. no data is sent and instead they are interpolated from the reference frames either side of the B slice (Chapter 6). One macroblock is selected near the centre of the frame. This is coded in B_8 × 8 mode, i.e. as four 8 × 8 macroblock partitions. Each partition is coded using Direct prediction (Chapter 6). The CBP (coded block pattern) map indicates that 5 of the 4 × 4 luma blocks contain coded coefficients, the other blocks being all zero.

9.3 Performance comparisons

9.3.1 Performance criteria

The H.264 standard defines a syntax and decoding method for video but does not specify how video should be encoded. In practice the constraints of the standard place certain limitations on encoder design and so most H.264 encoders share a common basic design (Chapter 4). However, there is scope for a wide range of performance, especially in terms of the following performance criteria:

Criterion	Description	Desirable
Total bitrate	Bitrate of complete compressed video sequence, bits per second.	Low bitrate at a given image quality, image resolution and frame rate.
Quality	Decoded image quality or distortion.	High quality/low distortion at a given bitrate, resolution and frame rate.
Processing rate	Speed of encoding or time taken to encode a sequence.	High processing rate, i.e. rapid encoding of a video sequence.
Bitrate control	Bitrate of coded sequence at a particular point in time.	

9.3.2 Performance examples: Foreman sequence, QCIF resolution

The following performance results were obtained by encoding 100 frames of the 'Foreman' video clip (Figure 9.6), QCIF resolution, 30 frames per second, using the JM reference encoder. By encoding the same sequence using a range of coding parameters, it is possible to explore the trade-offs between bitrate, quality and computational complexity. Each sequence is coded starting with an I slice, followed by P and optionally B slices. Note that a different H.264 encoder would be expected to produce different results, unless the algorithms by which the encoder selects coding options are identical.

9.3.2.1 'Low complexity' and 'Basic'

First we compare two configurations that may be suitable for devices with limited computational and storage capacity (Table 9.4). The parameters to be varied are as follows:

- Number of reference frames, previously coded frames used for inter prediction
- Minimum motion compensation block size, e.g. '8 × 8' means that the encoder may use any MC block size of 8 × 8 or larger
- Entropy coding, CAVLC or CABAC
- Loop filter, the built-in deblocking filter, switched on or off
- B slices, bipredicted slices between pairs of P slices
- Rate distortion optimization, i.e. repeatedly coding the macroblock in different modes and choosing the 'best' mode
- Rate control, i.e. varying the QP dynamically to meet a target bitrate.

Table 9.4 'Low Complexity' and 'Basic' configurations

Configuration	Low complexity	Basic
Number of reference frames	1	1
Smallest motion compensation block size	8 × 8	8 × 8
Entropy coding	CAVLC	CAVLC
Loop filter	Off	On
B slices	None	None
Rate Distortion Optimization	Off	Off
Rate control	Off	Off

Figure 9.9 shows the rate-distortion performance of these two configurations. To generate each graph, the sequence is coded at a range of QP values, in this case QP24, the top-right point on the graph, to QP36, the lower-left point, and the coded bitrate (kilo bits per second), luma PSNR (dB) and coding time (seconds) are recorded. Figure 9.9 shows that the 'Basic' configuration delivers better rate-distortion performance than the 'Low complexity' configuration, i.e. a higher quality at the same bitrate. Figure 9.13 charts the coding time of each sequence at one QP setting. The 'Basic' configuration takes only slightly longer to code, implying that the rate-distortion improvement comes at a small penalty in computational complexity.

9.3.2.2 'Basic' configuration plus options

Starting with the 'Basic' configuration described earlier, we add a number of individual coding options (Table 9.5) and measure the performance of (i) Basic + 1 option and (ii) Basic + the

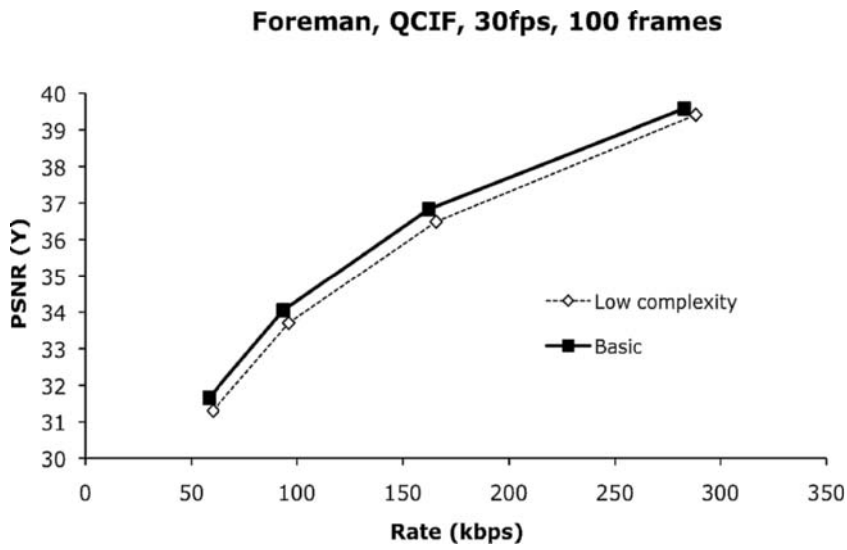


Figure 9.9 Foreman / QCIF / Basic complexity

Table 9.5 'Basic' plus options

Configuration	Basic	Basic +4 × 4	Basic + 5 ref	Basic +CABAC	Basic + 1B	Basic + RDO	Best Baseline
Number of reference frames	1	1	**5**	1	1	1	5
Smallest motion compensation block size	8 × 8	**4 × 4**	8 × 8	8 × 8	8 × 8	8 × 8	4 × 4
Entropy coding	CAVLC	CAVLC	CAVLC	**CABAC**	CAVLC	CAVLC	**CAVLC**
Loop filter	On	On	On	On	On	On	**On**
B slices	None	None	None	None	**One**	None	**None**
Rate Distortion Optimization	Off	Off	Off	Off	Off	**On**	**On**
Rate control	Off	Off	Off	Off	Off	Off	**Off**

'best' selection of options that are compatible with Baseline profile. Note that CABAC and B slices are not allowed in a Baseline Profile bitstream.

Comparing the performance of these options (Figure 9.10), it is clear that each individual option makes a small improvement to rate-distortion performance. Combining several options - multiple reference frames, 4 × 4 minimum block size and Rate Distortion Optimization - gives a more significant improvement in performance compared with the Basic configuration. However, this comes at a cost of a 4× increase in coding time (Figure 9.13).

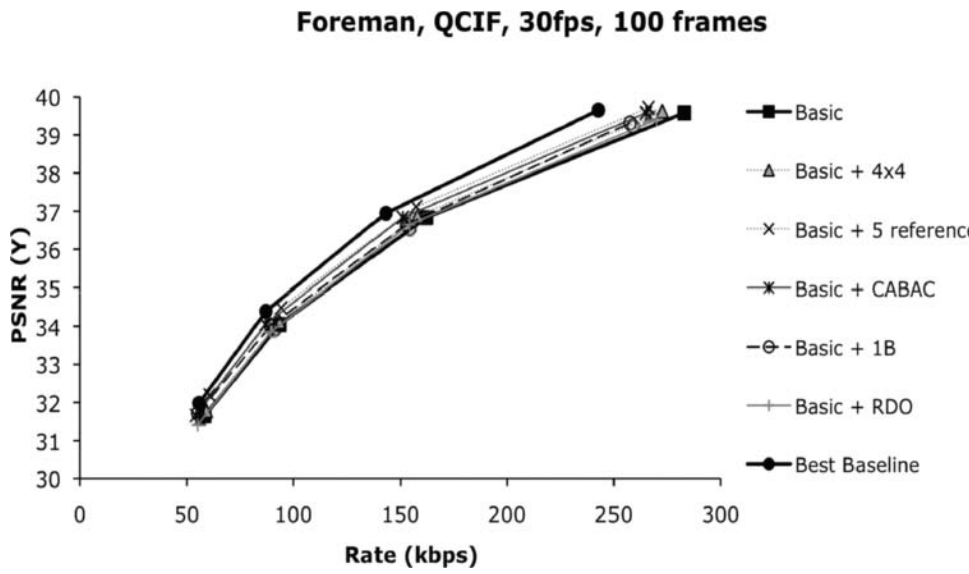


Figure 9.10 Foreman / QCIF / Basic complexity and options

Table 9.6 'Basic', 'Best Baseline', 'Medium'

Configuration	Basic	Best Baseline	Medium (1 ref)	Medium	Medium (rate control)
Number of reference frames	1	5	1	5	5
Smallest motion compensation block size	8 × 8	4 × 4	4 × 4	4 × 4	4 × 4
Entropy coding	CAVLC	CAVLC	CABAC	CABAC	CABAC
Loop filter	On	On	On	On	On
B slices	None	None	1	1	1
Rate Distortion Optimization	Off	On	On	On	On
Rate control	Off	Off	Off	Off	On

9.3.2.3 Baseline and Main Profile

Adding a selection of Main Profile tools gives the 'Medium Complexity' sequence. The new configuration is compared (Table 9.6) with the 'Basic' and 'Best Baseline' configurations. Adding CABAC and one B-slice between every two P-slices increases the performance of the 'Medium' sequence significantly (Figure 9.11). The coding time for the 'Medium' sequence is 28 seconds, compared with 21 seconds for the 'Best Baseline' sequence (Figure 9.13). For comparison, the 'Medium' sequence with only one reference frame is faster to encode but the performance drops as a result.

Many applications of video coding require a constant, or at least a constrained, bitrate. A rate control algorithm controls the QP in order to maintain an approximately constant coded

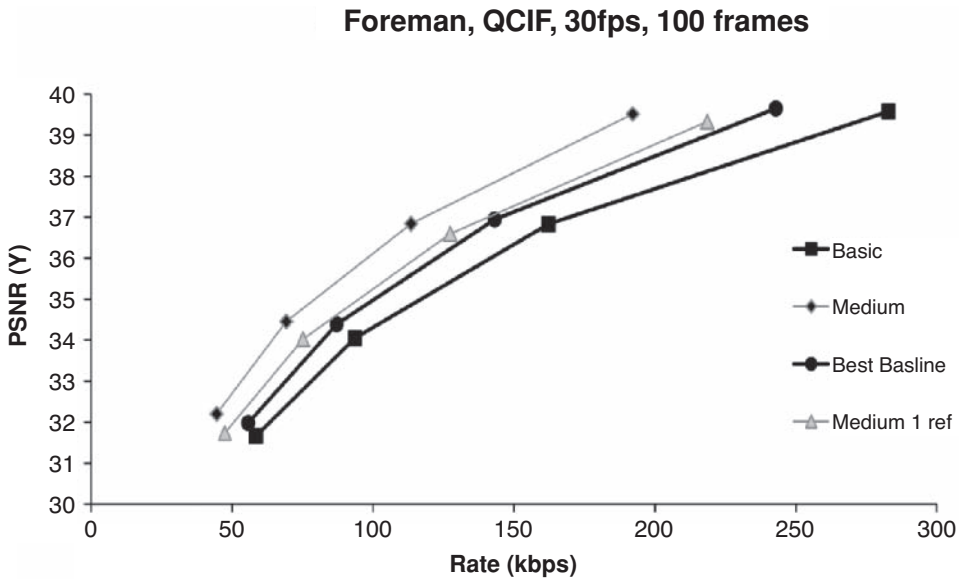


Figure 9.11 Foreman / QCIF / Basic and Medium Complexity

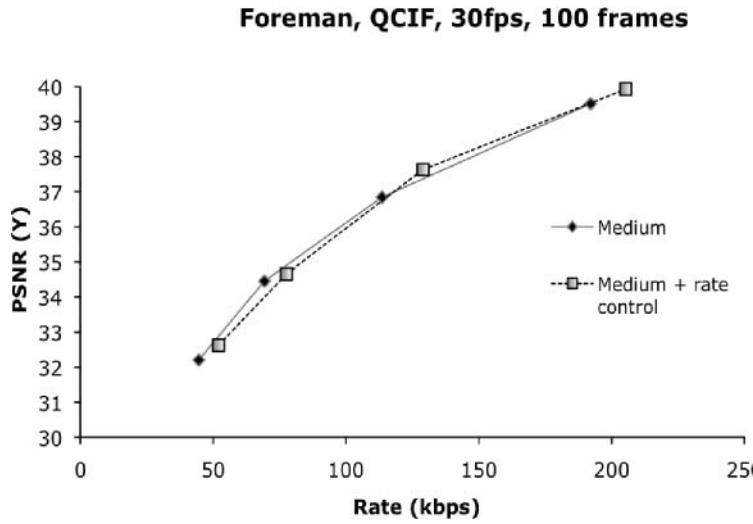


Figure 9.12 Foreman / QCIF / Medium complexity with rate control

bitrate, at least across a number of frames. Adding rate control increases the coding time slightly but does not significantly affect rate-distortion performance (Figure 9.12).

9.3.3 Performance examples: Foreman and Container sequences

An H.264 codec will perform differently depending on the content of the video sequence. For example, a sequence containing more motion and detail will tend to generate a larger number of bits than a sequence containing less motion and detail at a similar quality level.

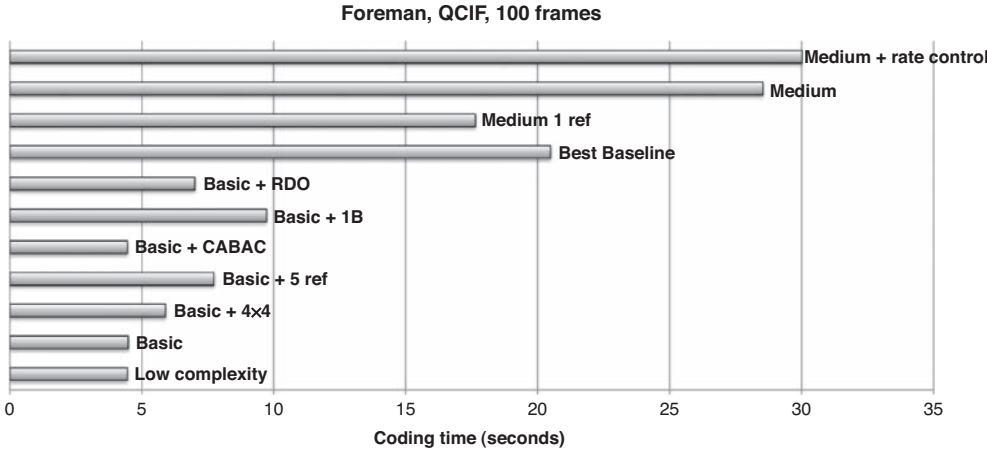


Figure 9.13 Foreman, QCIF sequences: coding time

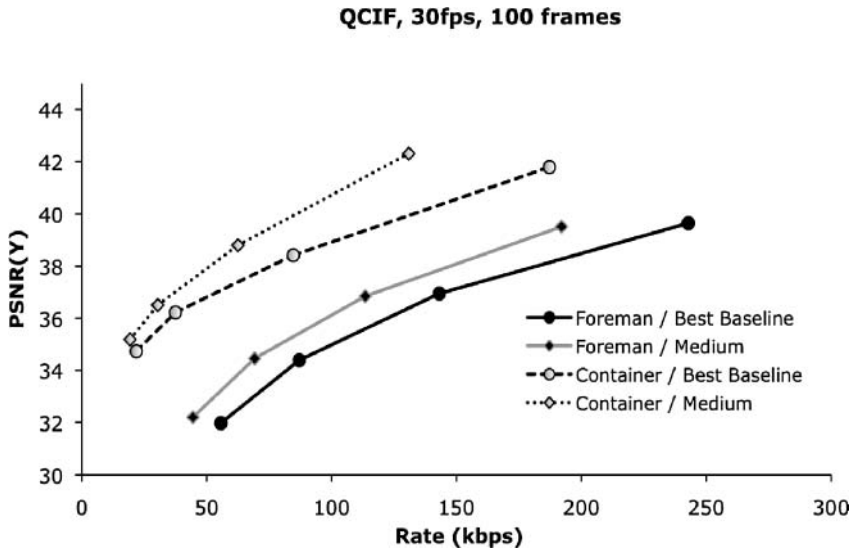


Figure 9.14 QCIF sequences: rate vs. PSNR

Figure 9.14 compares the rate-distortion performance of the Foreman and Container QCIF sequences using selected configuration settings. 'Container' (Figure 9.5) has less detail and less complex motion than 'Foreman'. At the same PSNR level and using the same configuration settings, 'Foreman' requires a bitrate three to four times higher than 'Container'.

The same sequences, Foreman and Container, are coded in CIF resolution. The larger source image size results in higher bitrates (Figure 9.15). Once again, Foreman requires more bits to code than Container at the same quality level, but the separation between the sequences is

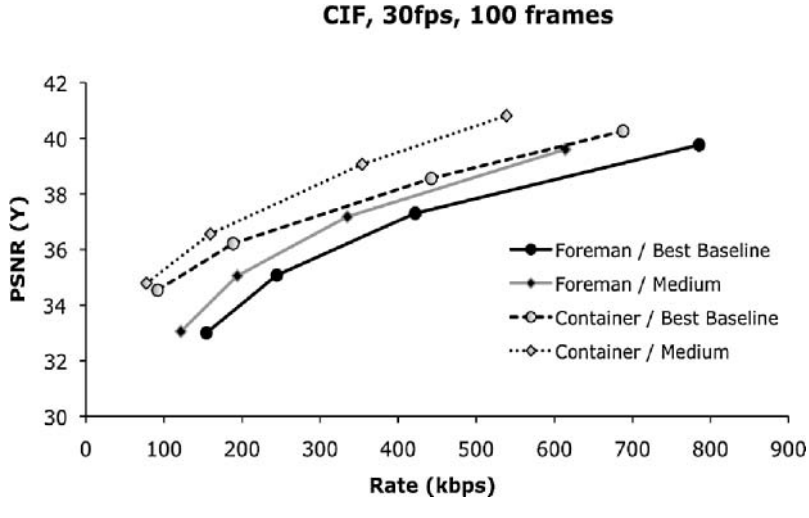


Figure 9.15 CIF sequences: rate vs. PSNR

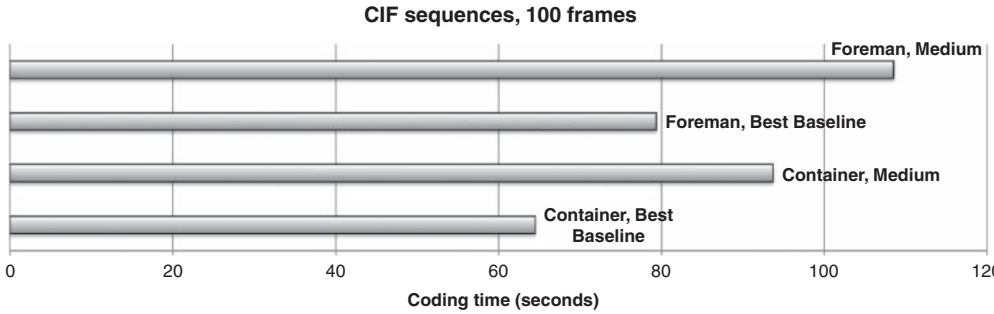


Figure 9.16 CIF sequences: coding time

smaller at CIF resolution than at QCIF resolution. The more complex Foreman sequence tends to take longer to encode (Figure 9.16) than Container.

9.3.4 Performance examples: Inter prediction structures

Figure 9.17 compares the compression performance of 'Foreman' coded using the four prediction structures described in Chapter 6. Each sequence is coded with the following common parameters:

- 61 frames of the 'Foreman' CIF test sequence
- Encoded using the JM reference software encoder, version 16.0
- Every 12th frame is coded as an I slice
- Main Profile, CABAC entropy coding
- Rate Distortion Optimised mode selection enabled
- No rate control.

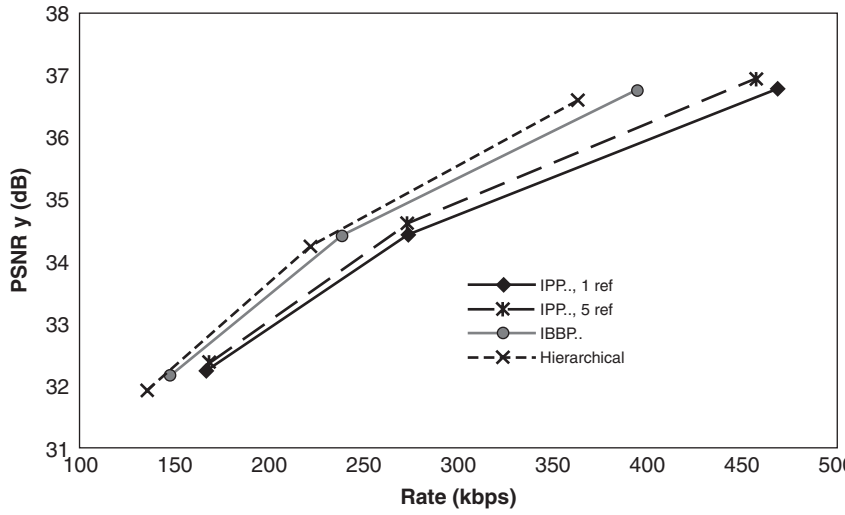


Figure 9.17 Rate-distortion comparison of prediction structures

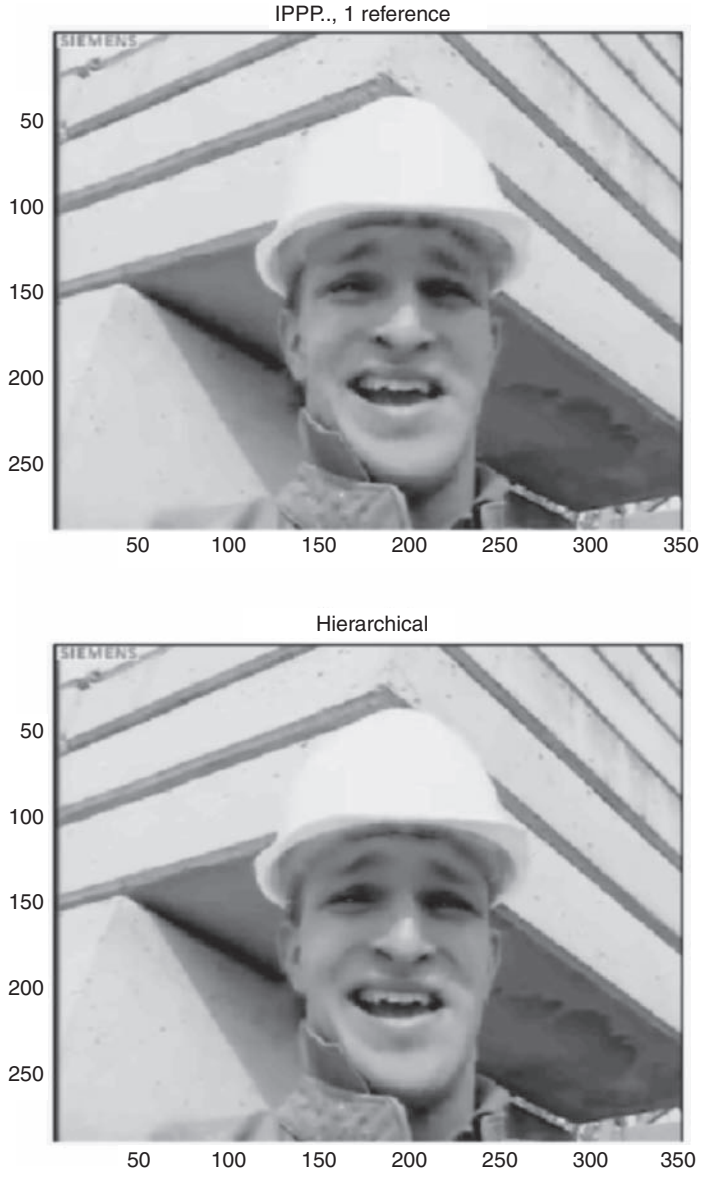


Figure 9.18 Sample frames from sequences using different prediction structures, coded at 280kbps

The prediction structures are as follows (see Chapter 6):

(i) IPPPPPPPPPPPIPP. . ., i.e. one I-slice followed by 11 P-slices, with one reference frame
(ii) As (i) but with five reference frames

(iii) IBBPBBPBBPBBIBBP. . ., i.e. 12-frame GOP with one I-slice, three P-slices and eight B-slices in each GOP.
(iv) IBBBBBBBBBBBIBB.., 12-frame GOP, hierarchical prediction.

The basic IPPPP. . . structure with one reference frame has the worst performance, i.e. the lowest rate-distortion curve. Enabling five reference frames with the same structure improves the performance slightly. The 'Classic' IBBPBBP. . . GOP structure, with 12 pictures in a GOP, improves the compression performance further and a Hierarchical GOP structure, with 12 pictures, gives the best performance. Sample frames from IPPP. . . and Hierarchical structures are shown in Figure 9.18, each coded at the same bitrate (280kbps). The luminance PSNR of the Hierarchical sequence is around 0.8dB higher than that of the IPPP. . . sequence; note that there is only a slight subjective difference in the frames. The Hierarchical sequence takes approximately 50 per cent longer to encode using the JM software.

9.3.5 Performance example: H.264 vs. MPEG-4 Visual

Figure 9.19 compares the rate-distortion performance of H.264 and the earlier MPEG-4 Part 2 ('Visual') standard. Similarly to H.264, MPEG-4 Visual has Profiles that define subsets of coding tools. Results for the 'Carphone' QCIF sequence coded using Simple Profile (SP) and Advanced Simple Profile (ASP) are shown. The SP encoder uses I- and P-frame coding and the ASP encoder adds B-frame coding for better performance. An H.264 Baseline Profile encoder using UVLC/CAVLC and one reference frame performs considerably better than MPEG-4 Visual ASP and H.264 Main Profile with CABAC and five reference frames performs better

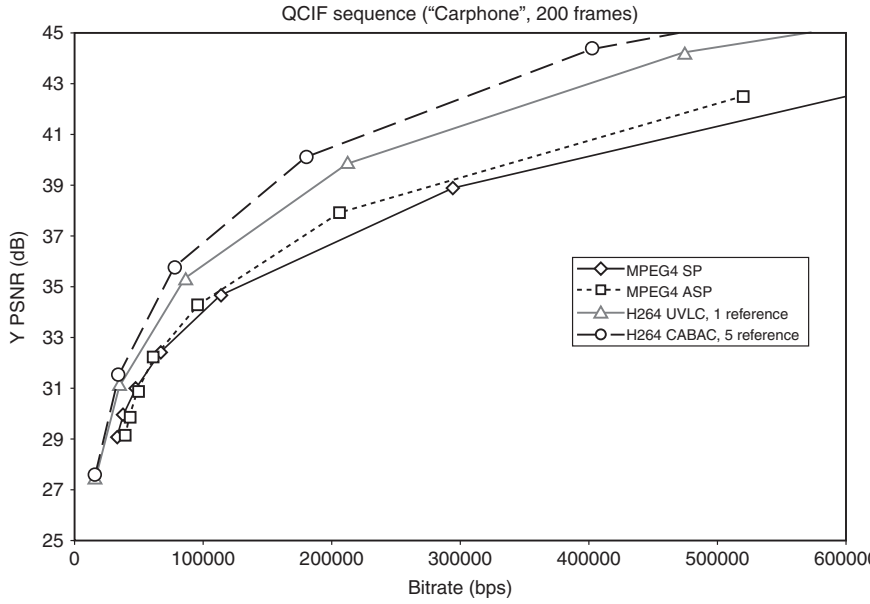


Figure 9.19 Carphone, QCIF: H.264 vs. MPEG-4 Visual

Figure 9.20 Frame from 'Foreman' sequence showing macroblock sizes

still, with a bitrate reduction of around 30-40 per cent compared with MPEG-4 ASP at a similar bitrate.

9.4 Rate control

The number of bits produced when an encoder codes a macroblock is not constant. For example, Figure 9.20 plots the number of bits per macroblock in a frame of 'Foreman' coded as a P slice. Lighter blocks are MBs with more bits, darker blocks contain fewer bits. Typically, more bits are required to code MBs that contain significant movement and/or detail, since these contain non-zero motion vector differences and non-zero transform coefficients.

In a similar way, the number of bits per coded frame is not constant. If all encoding parameters are kept constant, variations in motion and detail cause the bitrate to vary; for example, see the bitrate graph in Figure 9.7.

Practical applications of H.264/AVC require a constant bitrate output, or at least a constrained bitrate output. Some examples are listed in Table 9.7.

A typical coding scenario is shown in Figure 8.6 (Chapter 8). The encoder output buffer has a 'smoothing' or averaging effect on the coded bitrate. However, the constraints of the HRD (Chapter 8) mean that it is always necessary to control or manage the coded bitrate, unless the decoder can cope with an arbitrarily long decoding delay.

Table 9.7 Bitrate and delay constraints

Application	Bitrate and delay constraints
Video broadcast over fixed bitrate channel	Constant bitrate, medium delay
IP video streaming	Variable bitrate within limits, medium delay
IP videoconferencing	Variable bitrate within limits, low delay
DVD recording	Variable bitrate within limits, medium delay, fixed maximum file size

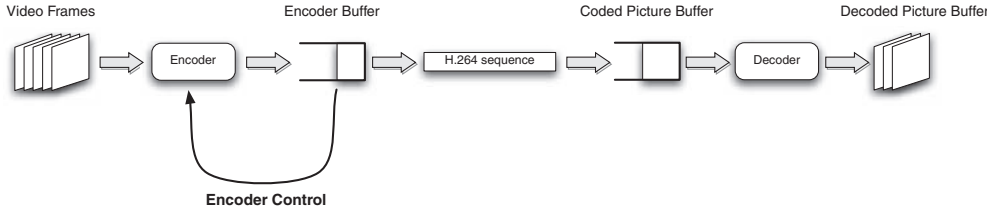


Figure 9.21 Encoder with rate feedback

Controlling the output bitrate is typically achieved by measuring the rate and/or the encoder buffer fullness level and feeding this back to control the encoder (Figure 9.21). Many of the encoder parameters can affect output bitrate e.g. type of slice, motion search range, mode selection algorithm, but the most useful parameter for bitrate control is the Quantizer Parameter (QP).

One way of controlling bitrate is simply to try and enforce a constant number of bits per coded frame, by measuring the output bitrate and feeding it back to control QP. Increasing QP reduces coded bitrate and decreasing QP increases coded bitrate. However, this approach is problematic because (i) it does not take into account the fact that coded I, P and B slices generate significantly different numbers of bits (Figure 9.8) and (ii) it will tend to lead to 'unpleasant' variations in image quality as the encoder increases or decreases QP rapidly to try and maintain bitrate.

A more flexible approach is outlined in Figure 9.22. The available channel bitrate, in bits per second, is used to determine a target number of bits for a Group of Pictures (GOP), typically an I slice followed a number of P and/or B slices. The bits available for the GOP are then

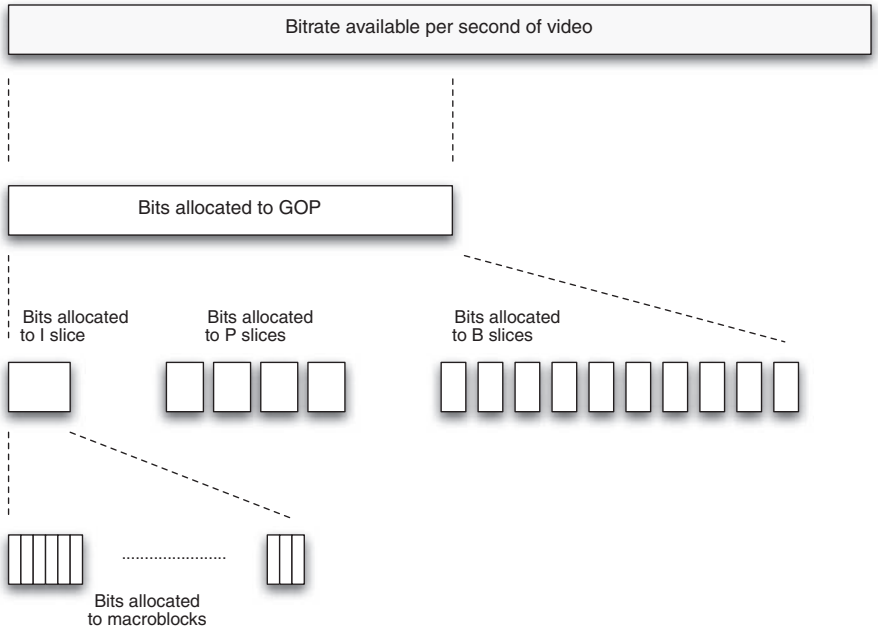


Figure 9.22 Bitrate allocation for rate control

allocated to I, P and B slices, with the allocation changing depending on the slice type. An I slice would typically be allocated most bits because intra prediction tends to be less efficient than inter prediction, followed by P slices and then B slices. Within each slice, a certain number of bits are allocated to each macroblock. The rate control algorithm then attempts to control the encoder to produce the target number of bits.

9.4.1 Rate control in the JM reference encoder

The rate control algorithm adopted in the JM reference encoder is described in its basic form in [iv] and [ix]. Various modifications have been proposed, some of which have been incorporated in later versions of the reference software [x]. The rate control algorithm attempts to (i) maintain a target coded bitrate during encoding and (ii) minimize obvious quality variations in the decoded video sequence. It operates with the following constraints:

Available bitrate R:	The number of bits per second, may be constant or variable.
Buffer size:	The size of the encoder output buffer and the decoder input buffer (CPB).
Video statistics:	The amount of motion and detail in the input video signal, typically varying.

The general approach is as follows.

1. Allocate a target number of bits for a **coded unit** based on:
   a. The available bitrate, taking into account the target bitrate and the actual bitrate produced up to this point.
   b. The buffer contents.
   c. The importance of the coded unit to future coding decisions, e.g. is it part of a slice that is used as a reference for further predicted frames.
2. Control the QP to match the target number of bits as closely as possible.
3. Update the parameters of the rate control algorithm based on the actual statistics of the input and coded sequences.

This approach is applied at various levels from GOP to coded picture and optionally down to the level of individual macroblocks or sequences of macroblocks. The minimum level at which rate is controlled is described as a **basic unit** and may be a single macroblock, a number of consecutive macroblocks or an entire coded frame.

9.4.1.1 GOP level rate control

Assume that the GOP structure consists of an I slice followed by P and/or B slices.

1. Calculate the number of bits available to code the GOP, based on:
   a. Available bitrate, constant or variable across the sequence, u
   b. Frame rate, F
   c. Number of frames in the GOP, N
   d. Size and occupancy of the encoder output buffer, B

2. Calculate the starting QP of the GOP based on:
   a. QPs allocated to frames in the previous GOP
   b. Target bitrate for previous and current GOPs.

The QP should not vary too much between GOPs in order to preserve reasonably consistent frame quality.

9.4.1.2 Frame and/or basic unit rate control

The following steps are applied once per frame if the basic unit is a complete coded frame, or multiple times if the basic unit is smaller than a frame.

1. Calculate a target number of bits for the coded frame.
2. Divide this number of bits equally amongst the basic units in the frame, may just be one basic unit.
3. Predict the Mean Absolute Difference (MAD) of the next basic unit.
4. Use a Rate-Distortion (R-D) model to estimate the QP required to produce the target number of bits for the basic unit, based on the estimated MAD of the basic unit.
5. Code the basic unit using this QP.
6. Update the parameters: number of bits available, MAD estimate, R-D model coefficients, based on the actual statistics of the coded basic unit.

Steps (3) and (4) require further explanation. The Mean Absolute Difference (MAD) is an estimate of the amount of 'activity', motion and/or detail, and therefore the likely size of the basic unit after coding. The relationship between MAD, QP and coded size of basic unit *i* can be modelled by the following quadratic expression (9.1):

$$T_i = c_1 \frac{MAD_i}{Qstep_i} + c_2 \frac{MAD_i}{Qstep_i^2} - h_i \tag{9.1}$$

Where T is the number of coded bits for the current basic unit, Qstep is the quantizer step size (related to QP, see Chapter 7), h is the number of bits required to code the header and motion vectors and c_1, c_2 are model coefficients, updated after coding each unit. Based on (9.1), the QP required to produce the correct number of coded bits T_i can be calculated. However, MAD is not known prior to coding the current basic unit. To get round this problem, MAD of the current basic unit *i* is estimated from the MAD of the basic unit in the same position (*co-located*) in the previous code frame (9.2). Coefficients a_1 and a_2 are updated after coding each basic unit.

$$MAD_i = a_1 MAD_{co\text{-}located} + a_2 \tag{9.2}$$

In this way, the encoder estimates the activity in the current basic unit and hence the quantizer parameter that is likely to produce the target number of bits T_i. The actual number of bits will vary from this, due to inaccuracies in the models. Based on the actual performance of the algorithm, the model parameters are updated in order to minimize the model error.

Example: Foreman, QCIF, 100 frames

100 frames of the Foreman QCIF sequence were encoded at a frame rate of 10 frames per second using Baseline profile, coded as one I-slice followed by P-slices. The rate control algorithm described in [ix] was used, with a target bit rate of 26 kbps. Figure 9.23 shows the coded bitrate. After the first (I) slice, the encoder maintains a roughly constant number of bits per frame.

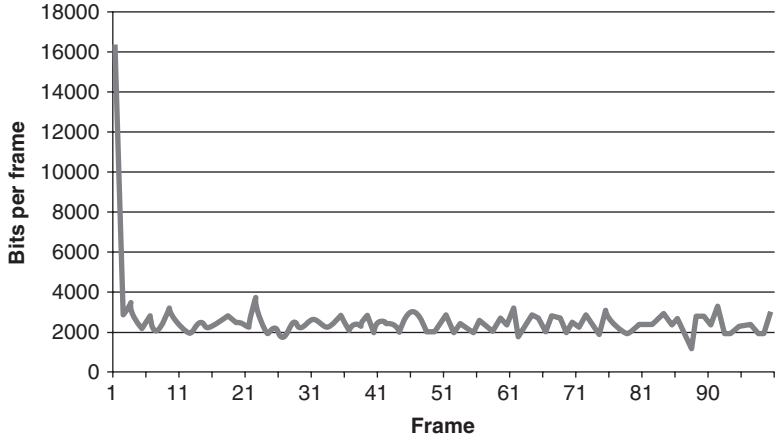


Figure 9.23 Foreman, QCIF, 100 frames: coded bitrate

'Foreman' is a ten-second clip that contains a relatively high amount of motion, particularly in the last 2–3 seconds. Figure 9.24 plots the variation of QP throughout the sequence. The large variation, particularly in the final seconds, is necessary in order to compensate for the changing motion and detail. This variation in QP leads to a variation in per-frame quality, measured as PSNR (Y) in Figure 9.25. As the QP increases, PSNR decreases and vice versa.

This example illustrates the classic trade-off of video codec rate control : a constant or near-constant bitrate typically is achieved at the expense of varying decoded quality.

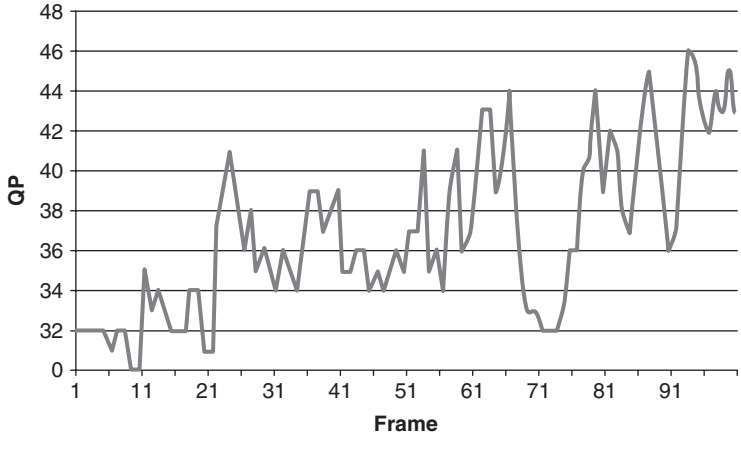


Figure 9.24 Foreman, QCIF, 100 frames: QP per frame

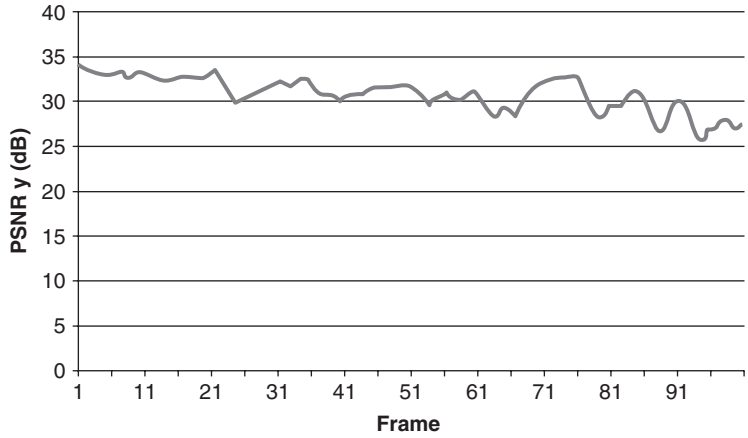


Figure 9.25 Foreman, QCIF, 100 frames: Luma PSNR per frame

9.5 Mode selection

An H.264/AVC encoder can choose from many different options or **modes** when it codes a macroblock. Figure 9.26 shows the main prediction choices for a macroblock; see Chapters 5 and 6 for more details. These include:

- 'Skip' mode: don't send any information for this macroblock
- Four intra-16 × 16 modes
- Nine intra-4 × 4 modes, with a different choice possible for each 4 × 4 block
- 16 × 16 inter mode with prediction from reference picture(s) from one (P, B MB) or two (B MB) lists
- 8 × 16 inter mode: prediction from multiple reference pictures as above, with the option of different reference picture(s) for each partition
- 16 × 8 inter mode with reference picture choices as above
- 8 × 8 inter mode with reference picture choices as above, with further sub-division of each 8 × 8 partition into 8 × 4, 4 × 8 or 4 × 4 sub macroblock partitions.

As well as the choice of prediction mode, the encoder can choose to change the quantization parameter (QP); within each inter mode the encoder has a wide choice of possible motion vectors; and so on. There are a huge number of options for coding each macroblock. Each coding mode, i.e. each combination of coding parameters, will tend to generate a different number of coded bits, ranging from very low (P-Skip or B-Skip) to high (Intra) and a different **distortion** or reconstructed quality.

A video encoder aims to minimize coded bitrate and maximise decoded quality or minimize decoded distortion. However, choosing the coding mode of a macroblock to achieve this is a difficult problem, because of (a) the large number of possible combinations of encoding parameters and (b) the question of deciding the 'best' tradeoff between minimizing bitrate and minimizing distortion.

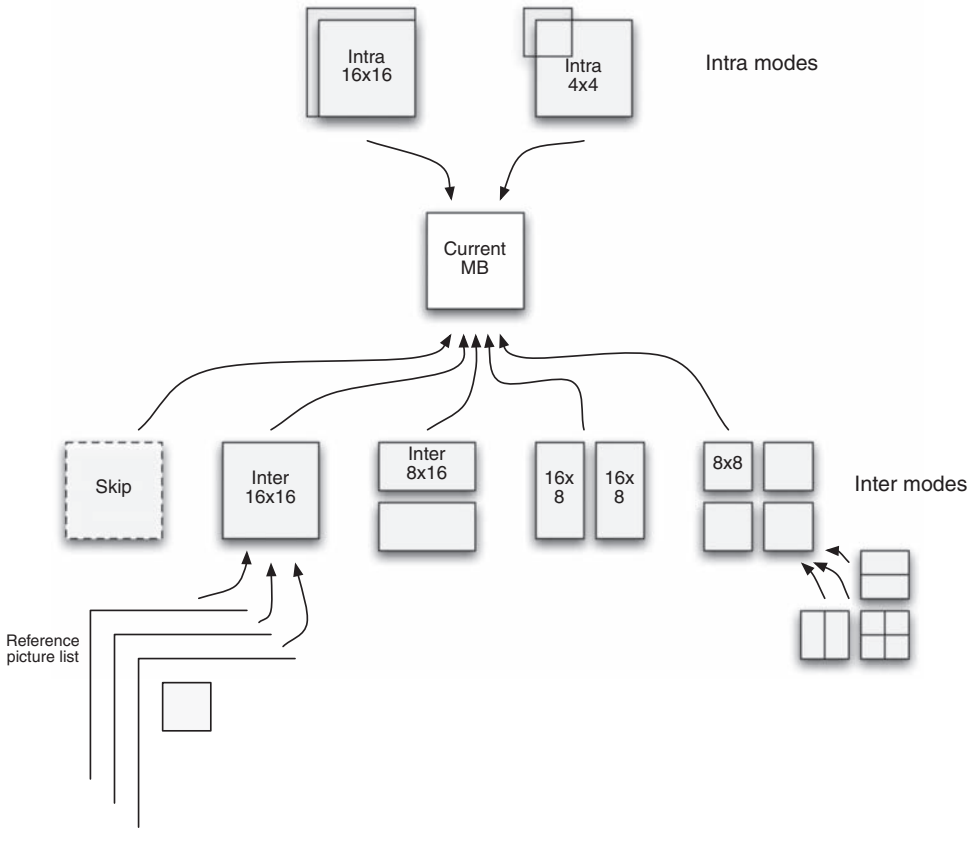


Figure 9.26 Available macroblock prediction modes

Example: 'Cost' of coding a macroblock

Figure 9.27 compares the rate and distortion costs of coding a particular macroblock in different modes. The three measurements are:

Header bits:	The number of bits required to signal the macroblock mode, plus any prediction parameters such as intra mode, reference choices and/or motion vector differences.
Coefficient bits:	The number of bits required to code the quantized transform coefficients.
MSE:	Distortion of the decoded, reconstructed macroblock, measured as Mean Squared Error.

'Skip' mode sends only a single bit to indicate that no further data is coded for this macroblock. The rate cost is negligible. However, the decoder must reconstruct the MB based on previously-coded data. If there are any significant changes from the previous decoded frame, the MSE is likely to be very high as in this case.

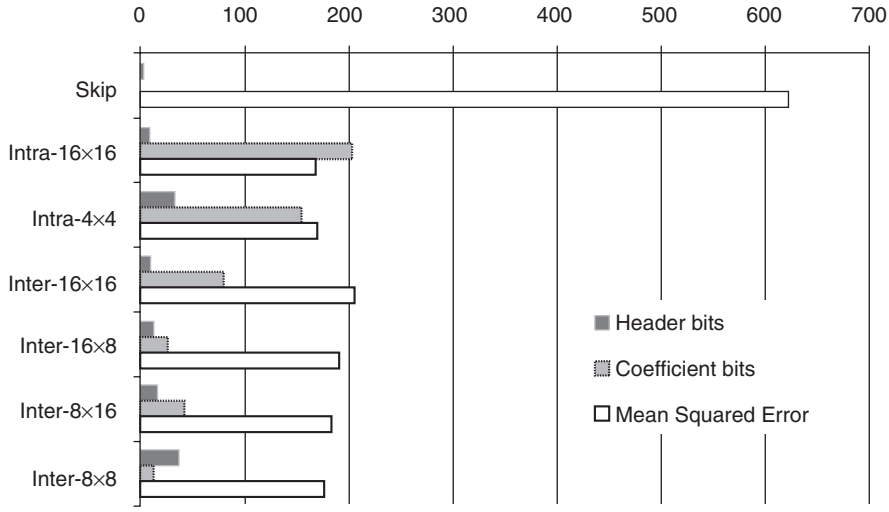


Figure 9.27 Rate and MSE costs for different coding options

The two Intra modes (16 × 16 and 4 × 4) give a lower MSE at the expense of a higher rate cost, particularly to signal the residual coefficients. Inter modes with larger block sizes e.g. 16 × 16 tend to have a low header cost but more coefficient bits, because the motion compensated prediction is not entirely accurate. Inter modes with smaller block sizes tend to have a larger header cost to signal mode and multiple motion vectors and fewer coefficient bits due to more accurate prediction.

The 'best' choice of mode depends on (i) the particular characteristics of the macroblock and (ii) the chosen weighting between distortion and rate. In this example, an encoder that is biased towards minimizing distortion, will tend to choose one of the Intra modes; an encoder that is biased towards minimizing rate, will tend to choose Skip mode.

9.5.1 Rate Distortion Optimized mode selection

Rate Distortion Optimized (RDO) mode selection is a technique for choosing the coding mode of a macroblock based on the rate and distortion cost. In its most popular formulation, the bitrate cost R and distortion cost D are combined into a single cost J (9.3):

$$J = D + \lambda R \tag{9.3}$$

The RDO mode selection algorithm attempts to find a mode that minimizes the joint cost J. The trade-off between Rate and Distortion is controlled by the Lagrange multiplier λ. A smaller λ will give more emphasis to minimizing D, allowing a higher rate, whereas a larger λ will tend to minimize R at the expense of a higher distortion. Selecting the best λ for a particular sequence is a highly complex problem [xi]. Fortunately, empirical approximations have been developed that provide an effective choice of λ in a practical mode selection scenario [xii].

Good results can be obtained by calculating λ as a function of QP (9.4, from [iv]).

$$\lambda = 0.852^{(QP-12)/3} \tag{9.4}$$

Distortion (D) is calculated as the Sum of Squared Distortion (SSD, (9.5):

$$D_{SSD} = \sum_{x,y} \left(b(x, y) - b'(x, y)\right)^2 \tag{9.5}$$

Where *x,y* are the sample positions in a block, *b(x,y)* are the original sample values and *b'(x,y)* are the decoded sample values at each sample position. Other distortion metrics, such as Sum of Absolute Differences or Sum of Absolute Transformed Differences, SAD and SATD, may be used in processes such as selecting the best motion vector for a block [iv]. A different distortion metric typically requires a different λ calculation.

A typical mode selection algorithm might proceed as follows:

- For every macroblock
  - For every available coding mode *m*
    - Code the macroblock using mode *m* and calculate *R*, the number of bits required to code the macroblock
    - Reconstruct the macroblock and calculate *D*, the distortion between the original and decoded macroblocks
    - Calculate the mode cost J_m using (9.3), with appropriate choice of λ
  - Choose the mode that gives the minimum J_m

This is clearly a computationally intensive process, since there are hundreds of possible mode combinations and therefore it is necessary to code the macroblock hundreds of times to find the 'best' mode in a rate-distortion sense. The problem becomes larger when considering the choice of prediction, for example:

1. Each 4 × 4 block in an Intra-4 × 4 macroblock can be coded using one of nine modes. The choice of mode for one 4 × 4 block affects the coding cost and therefore the choice of mode for all the other 4 × 4 blocks (see Chapter 6, Intra mode signalling).
2. Each partition in an Inter macroblock may be predicted from any one of the available reference frames or from one or two reference frames in a B macroblock.
3. Each partition or sub-macroblock partition in an Inter macroblock has a separate motion vector or two in the case of a bi-predicted partition in a B macroblock; each motion vector may point to any of hundreds of positions within a defined search window.
4. The choice of motion vector or prediction is further complicated by the fact that more likely predictions, e.g. small motion vectors or intra modes similar to recently-selected modes, generally require fewer bits to code than predictions such as large motion vectors that are less likely to generate the best result.

In the case of a B macroblock, for example, an encoder has a huge potential 'space' of coding options, including all the intra modes, all the inter partition sizes, all the possible motion vectors, all the available reference frames and the choice of one-directional or bi-predicted

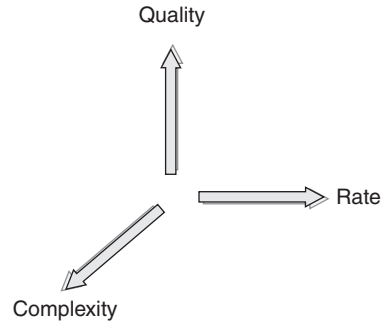


Figure 9.28 Rate, quality and complexity

motion compensation. Exhaustively searching this space to find the best combination of mode and prediction type is a highly computationally intensive task.

9.6 Low complexity coding

Many practical H.264/AVC codecs simply do not have the computational resources to carry out the full rate-distortion optimized mode selection process described above. This practical constraint, together with the desire to maximize compression performance, has led the development and proposal of hundreds of low complexity coding algorithms and approaches. The general goal is to maximize performance in the rate-distortion-complexity space (Figure 9.28). A low-complexity encoder will tend to have poor or average rate-distortion performance; applying more computation to the problem of choosing the best coding mode, hence moving along the complexity axis, will tend to increase rate-distortion performance. Hence compressed bitrate, decoded quality and codec complexity can be traded off. The goal of a good low-complexity coding method is to achieve good rate-distortion performance at a reduced computational cost. A suitable comparison point is a 'full complexity' H.264 encoder, which evaluates every possible coding mode, every prediction type and every motion vector when coding each macroblock.

In general, every low-complexity coding algorithm has a similar effect on performance, i.e. rate-distortion performance tends to be lower than the benchmark 'full complexity' codec and computational complexity is also lower. Some of the more sophisticated low-complexity algorithms, however, are capable of delivering coding performance very close to the benchmark with a significant reduction in computational complexity.

9.6.1 Approximating the cost function

The basic cost function (9.3) requires calculation of D and R. In a 'full' implementation this means it is necessary to:

(i) Code the block B to obtain R
(ii) Decode the block to obtain B', the reconstructed block
(iii) Calculate the distortion (e.g. SSD) between B and B'

A number of approximations to SSD have been proposed, with the aim of reducing the number of processing steps required to calculate distortion. For example, Sum of Absolute Differences (SAD) calculates the absolute difference between pairs of samples *b* and *b'* (9.6). SAD increases monotonically with SSD but is less computationally intensive to calculate. Calculating the absolute difference in the transform domain can improve the accuracy of the cost function approximation. (9.7) describes the transform-domain metric Sum of Absolute Transformed Differences (SATD), where T is a transform such as the Hadamard Transform and α is a normalization factor) [iv].

$$D_{SAD} = \sum_{x,y} \left| b(x, y) - b'(x, y) \right| \tag{9.6}$$

$$D_{SATD} = \alpha \sum_{x,y} \left| T\left(b(x, y) - b'(x, y) \right) \right| \tag{9.7}$$

Further fast approximations to these and other distortion metrics have been proposed in the literature, e.g. [xiii, xiv]. An alternative is to sub-sample the block, i.e. reduce the number of sample positions (x, y) to be evaluated [xvi, xv].

A simple approach to approximating the rate of a coded block is to predict blocks with zero rate, i.e. all-zero coefficients [xvii]. As all-zero blocks occur very frequently in a typical coded sequence, predicting the occurrence of these blocks without actually coding the data can save a significant amount of computation. A more sophisticated rate estimation model is presented in [xviii].

Finally, the entire cost function (9.3) may be estimated, for example based on the R-D cost of the same macroblock position in a previously coded frame [xix, xx].

9.6.2 Reducing the set of tested modes

A second approach to reducing the complexity of mode selection is to cut the number of modes that are tested for a given macroblock or block.

The 'Skip' mode tends to occur very frequently in P and B slices, especially when (i) the scene activity is relatively low and/or (ii) the quantization parameter is relatively high. Many methods have been developed that incorporate early skip detection based on modelling previous macroblock statistics, for example [xix, xxi, xx].

More generally, a number of algorithms attempt to reduce the cost of inter mode selection by grouping modes and only evaluating or coding certain groups of inter modes. Groupings can be determined by the statistics of previously coded macroblocks and/or according to homogeneity or features of the current macroblock [xxii, xxiii].

The number of intra modes tested for a MB / block may be reduced by examining the structure of the image data. The best intra mode tends to depend on the characteristics of the block or macroblock. For example, a macroblock containing smooth texture is likely to be effectively predicted using an Intra-16 $\times$ 16 mode (see Chapter 6) [xxiv]. Alternatively, the best Intra-4 $\times$ 4 mode for a particular 4 $\times$ 4 block is likely to be closely correlated to the block texture. For example, the dominant edge direction in a 4 $\times$ 4 block may be used to predict the most likely prediction direction [xxv].

9.6.3 Early termination

Related to the strategy of reducing the number of modes is the concept of early termination, which involves (i) evaluating coding modes in a certain order, which may be fixed or variable, and (ii) terminating the process when certain criteria are reached. For example, a number of algorithms assume that the costs of certain inter modes are monotonically increasing or decreasing, i.e.:

$$J_{16\times16} > J_{8\times8} > J_{4\times4}$$

or

$$J_{16\times16} < J_{8\times8} < J_{4\times4}$$

Evaluating the modes in order of expected monotonicity, the encoder terminates the mode selection process if a particular mode cost is not monotonic [xxvi, xxvii].

9.7 Summary

A good way to understand and evaluate the capabilities of a video coding method such as H.264/AVC is to experiment with it. The availability of public-domain encoders and decoders such as the 'official' Joint Model (JM) and the open-source ×264, makes it possible to test out every aspect of the standard. H.264/AVC has the potential to deliver coding performance as good as, or better than, other standards-based and proprietary codecs that are currently available. However, the performance of an H.264 codec depends very much on the coding parameters and the source video material. There is a fundamental trade-off between good coding performance - high quality and low bitrate - and computational complexity, particularly in a video encoder. A key aspect of this trade-off is the challenge of selecting the best coding mode for a macroblock, out of a very large number of possible options. Practical H.264/AVC encoders typically simplify this process using fast, low-complexity approximations to the full mode selection process.

9.8 References

i. JM reference software version 16.0, http://iphome.hhi.de/suehring/tml/, July 2009.
ii. ITU-T Recommendation H.264.2 : Reference software for H.264 advanced video coding, June 2008.
iii. A. M. Tourapis, A. Leontaris, K. Suehring and G. Sullivan, 'H.264/MPEG-4 AVC Reference Software Manual', Joint Video Team Document JVT-AD010, January 2009.
iv. K. P. Lim, G. Sullivan and T. Wiegand, 'Text Description of Joint Model Reference Encoding Methods and Decoding Concealment Methods', Joint Video Team Document JVT-O079, April 2005.
v. G. Sullivan, 'Adaptive quantization encoding technique using an equal expected-value rule', Joint Video Team Document JVT-N011, Jan 2005.
vi. × 264 public domain H.264 encoder/decoder, http://www.videolan.org/developers/× 264.html.
vii. Doom9 Codec Shoot-Out 2005, http://www.doom9.org/index.html?/codecs-final-105-1.htm, December 2005.
viii. Fourth Annual MSU MPEG-4 AVC/H.264 Video Codec Comparison, http://compression.ru/video/codec_comparison/mpeg-4_avc_h264_2007_en.html, December 2007.
ix. Z. Li, F. Pan, K. P. Lim, G. Feng, X. Lin and S. Rahardja, 'Adaptive Basic Unit Layer Rate Control for JVT', Joint Video Team Document JVT-G012, March 2003.

x. A. Leontaris and A. M. Tourapis, 'Rate Control reorganization in the Joint Model (JM) reference software', Joint Video Team Document JVT-W042, April 2007.
xi. A. Ortega and K. Ramchandran, 'Rate-distortion methods for image and video compression', *IEEE Signal Processing Magazine*, November 1998.
xii. G. Sullivan and T. Wiegand, 'Rate-distortion optimization for video compression', *IEEE Signal Processing Magazine*, November 1998.
xiii. C. Tseng, H. Wang and J. Yang, 'Enhanced intra-4 × 4 mode decision for H.264/AVC coders', *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 16, no. 8, August 2006, pp. 1027–1032.
xiv. J. Xin, A. Vetro and H. Sun, 'Efficient macroblock coding-mode decision for H.264/AVC video coding', Proceedings of the Picture Coding Symposium, 2004.
xv. Y. Ivanov and C. Bleakley, 'Skip prediction and early termination for fast mode decision in H.264/AVC', Proceedings of the International Conference on Digital Telecommunications, 2006.
xvi. Y. Huang, B. Hsieh, T. Chen and L. Chen, 'Analysis, fast algorithm and VLSI architecture design for H.264/AVC intra frame coder', *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 15, no. 3, pp. 378–401, March 2005.
xvii. Y. Moon, G. Kim and J. Kim, 'An improved early detection algorithm for all-zero blocks in H.264 video encoding', *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 15, no. 8, pp. 1053–1057, August 2005.
xviii. H. Kim and Y. Altunbasak, 'Low complexity macroblock mode selection for H.264/AVC encoders', Proc. IEEE International Conference on Image Processing, October 2004.
xix. C. Kannangara, I. Richardson, M. Bystrom, J. Solera, Y. Zhao, A. MacLennan and R. Cooney, 'Low complexity skip prediction for H.264 through Lagrangian cost estimation', *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 16 no. 2, pp. 202–208, February 2006.
xx. M. Bystrom, I. Richardson and Y. Zhao, 'Efficient mode selection for H.264 complexity reduction in a Bayesian framework', *Signal Processing: Image Communication*, vol. 23, no. 2, pp. 71–86, February 2008.
xxi. B Jeon and J Lee, 'Fast mode decision for H.264', Joint Video Team document JVT-J033, ISO/IEC JTC1/SC29/WG11 and ITU-T SG16 Q.6, Hawaii, December 2003.
xxii. P Yin, H Tourapis, A Tourapis and J Boyce, 'Fast mode decision and motion estimation for JVT-H.264', Proc. IEEE International Conference on Image Processing, September 2003.
xxiii. A.C. Yu, 'Efficient Block-Size Selection Algorithm for Inter-Frame Coding in H.264/MPEG-4 AVC,' Proc. IEEE International Conference on Acoustic, Speech, Signal Processing, May 2004.
xxiv. C. Yang, L. Po and W. Lam, 'A fast H.264 intra prediction algorithm using macroblock properties', Proc. IEEE International Conference on Image Processing, October 2004.
xxv. F. Pan, X. Lin, S. Rahardja, K. Lim, Z. Li, D. Wu and S. Wu, 'Fast mode decision algorithm for intra prediction in H.264/AVC video coding', IEEE Transactions on Circuits and Systems for Video Technology, vol. 15, no. 7, pp. 813–822, July 2005.
xxvi. Z. Zhou, M. Sun, and Y. Hsu, 'Fast Variable Block-Size Motion Estimation Algorithms Based on Merge and Split Procedures for H.264/MPEG-4 AVC,' Proc. International Symposium on Circuits and Systems, 2004.
xxvii. L. Salgado and M. Nieto, 'Sequence Independent Very Fast Mode Decision Algorithm on H.264/AVC Baseline Profile,' Proc. IEEE International Conference on Image Processing, 2006.

10

Extensions and directions

10.1 Introduction

Since the first version of H.264/AVC was published in 2003, the video coding industry has continued to evolve. The range of platforms and delivery mechanisms for video continues to grow, with an increasing expectation that video content should be available on any platform from mobile to HD and 3D displays, over any network including broadcast, internet, mobile, etc. The standard itself has evolved since 2003. This chapter summarizes recent extensions to the standard and looks at what might come after H.264.

The so-called 'Professional' or 'Fidelity Range' extensions became the High Profiles of H.264, tools for coding High Definition and studio content with very high reproduction fidelity, described in earlier chapters.

The increasing need for coding the same original content at different bandwidths and display resolutions led to the development of the Scalable Video Coding (SVC) extension to H.264, standardised as H.264 SVC. SVC supports efficient coding of video in such a way that multiple versions of the video signal can be decoded at a range of bitrates, spatial resolutions and/or temporal resolutions or frame rates. By jointly coding multiple versions, it should be possible to deliver them in a more efficient way than the alternative of coding and transmitting each version separately.

There is a trend towards creating and delivering multiple views of the same video scene. Stereoscopic video, with suitable display technology, gives the impression of a three-dimensional (3D) image. Multiple views of a scene can give the user the option of choosing their viewpoint. 'Free viewpoint' video can potentially deliver any view of a scene, by synthesising intermediate views between actual camera positions. These 'multiview' applications generally require coding of multiple, closely related video signals or views. Similarly to SVC, Multiview Video Coding (MVC) exploits the correlation between these views to deliver efficient compression. Tools for multiview video coding have been standardized as H.264 MVC.

The number of video compression formats continues to increase, with more and more content being produced and coded into many different, incompatible compression formats. Recent initiatives in configurable video coding address the problem of efficiently supporting an increasing range of compression formats. MPEG's Reconfigurable Video Coding (RVC)

The H.264 Advanced Video Compression Standard, Second Edition Iain E. Richardson
© 2010 John Wiley Sons, Ltd

sub-group has defined a Video Tool Library, a set of standard Functional Units, building blocks for video compression. A particular video codec may be specified by defining a subset of Functional Units together with their parameters and interconnections. This should make it possible to flexibly re-configure a video codec to support multiple standard and proprietary formats. Going further, Fully Configurable Video Coding (FCVC) makes it possible to completely define and implement an arbitrary video codec using a set of low-level primitive operations and to change this definition during a video communication session. Potential benefits of this approach include the ability to rapidly implement any new video coding algorithm and to adapt the compression algorithm dynamically to suit the characteristics of the current video scene.

H.264 has proved to be a useful and successful technical standard and continues to increase its share of the video coding market. As processor capabilities continue to develop, the standards community are considering what should come after H.264/AVC. At the time of writing (early 2010), the MPEG and VCEG groups are examining proposals for a next generation video coding standard that is expected to offer better performance than H.264, probably at a higher computational cost.

10.2 Scalable Video Coding

10.2.1 Simulcast transmission

A challenge for many video compression applications is to deliver multiple versions of a video sequence at different operational points, i.e. different qualities, spatial resolutions and frame rates. This can be done using conventional video codecs such as H.264/AVC by coding each stream independently. This is **simulcast**. In a typical scenario (Figure 10.1), a single source video is required to be transmitted to multiple decoders or clients, each with different capabilities. In this example, the original video clip is coded three times to produce three independent AVC streams, each of which is transmitted and decoded. The problem with the simulcast scenario is that the three bitstreams contain significant redundancy, since the same video sequence is coded in each bitstream at different resolutions and/or qualities. In theory, a

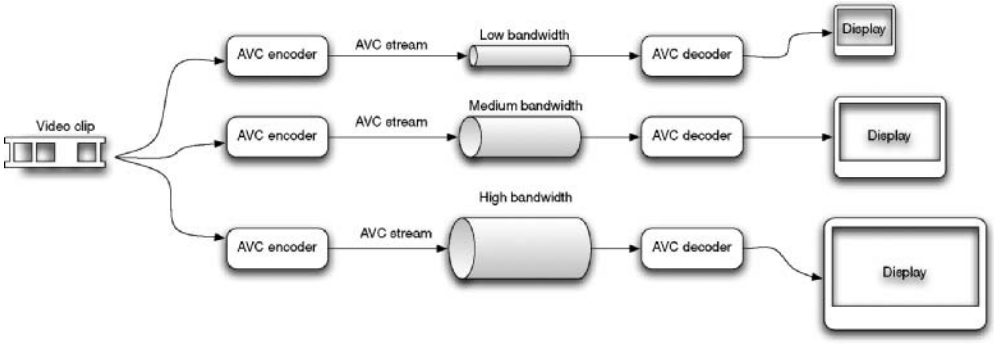


Figure 10.1 Multiple streams / simulcast

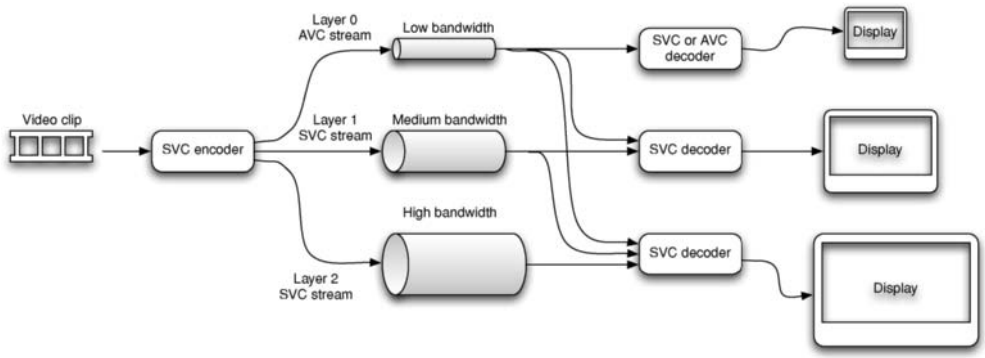


Figure 10.2 Multiple streams / scalable

smaller transmission bandwidth could be utilized by exploiting this redundancy between the three streams.

10.2.2 Scalable transmission

Scalable Video Coding (SVC) attempts to deliver multiple coded versions of a sequence using a lower overall bitrate than the simulcast scenario above. It does this by exploiting the redundancies between the different versions, i.e. the correlation between different versions of the same sequence coded at different operating points.

The same three sequences delivered using SVC are shown in Figure 10.2. A single SVC encoder produces three coded bitstreams, described as **layers**. The lowest or **base layer**, layer 0 in the figure, is a stream decodeable using a standard single-layer decoder, e.g. an H.264 decoder, to produce a video sequence at the lowest of the available quality/resolution operating points. One or more **enhancement layers**, layers 1 and 2 in this example, are coded as SVC bitstreams. To decode a sequence at a higher quality or resolution, an SVC decoder decodes the base layer **and** one or more enhancement layers. In this example, decoding layer 0 using a standard AVC decoder produces the lowest quality output; decoding layers 0 and 1 using an SVC decoder produces a higher quality output; decoding layers 0, 1 and 2 using an SVC decoder produces the highest-quality output. The SVC coding process exploits redundancy between sequences coded at different resolutions or qualities, by predicting successive enhancement layers from the base layer and lower enhancement layers. In this way, it should be possible to achieve the same displayed result as the simulcast system (Figure 10.1) at a reduced bandwidth cost.

The general concept of a scalable coded bitstream is that 'parts of the stream can be removed in such a way that the resulting sub-stream forms another valid bit stream for some target decoder' [i]. Considering Figure 10.2, the scalable bitstream consists of the coded Layer 0, Layer 1 and Layer 2 streams. Decoding all three streams produces a high-quality output; removing Layer 2 and decoding layers 0 and 1 produces a medium-quality output; removing layers 1 and 2 and decoding just the base layer produces a low-quality output.

10.2.3 Applications of Scalable Video Coding

Scalable video coding has been proposed for a number of application scenarios.

Multiple decoders: Increasingly, the same original video content is coded, transmitted and viewed by multiple devices, each with different capabilities. For example, a movie trailer is streamed to clients ranging from a handheld device with a low bitrate network connection and a low-resolution display, to a PC with a high bitrate connection and a High Definition display. A range of factors may limit the capabilities of a particular decoding device, including connection bitrate, screen resolution and processing capacity. A scalable bitstream should make it possible to support a wide range of decoding capabilities as efficiently as possible.

Graceful degradation / enhancement: Whilst some applications such as broadcast television tend to have a clearly-defined and consistent channel for video transmission, many other applications use a channel that may change significantly during a communication session. For example, IP-based applications such as video streaming or internet conferencing will experience a varying channel throughput that depends on factors such as the amount of traffic and congestion in the network. Scalable coding offers a mechanism for maximising the quality at a particular point in time for a specific decoder. For example, a streaming server transmits base and enhancement layers for a video source. A decoder attempts to receive each of the available layers. If all layers are successfully received, the decoder extracts a sequence at the maximum available quality. If the connection throughput drops, the decoder 'drops back' to a lower-quality sequence by only receiving selected layers. As long as the base layer is successfully decoded, a basic-quality video sequence can be displayed at all times. This implies that the base layer is very important, i.e. a higher priority than the enhancement layer(s).

Archiving: Storing a video sequence as a scalable coded bitstream can make it possible to rapidly decode a low-quality 'preview' of the video sequence. For example, a HD sequence is coded as a number of scalable layers. Extracting only the base layer gives a low quality version that is quick to decode and display, suitable as a preview of the entire sequence.

10.2.4 Scalable video coding in H.264

Scalable Video Coding (SVC) is incorporated as Annex G of recent versions of the H.264/AVC standard [i,ii] and extends the capabilities of the original standard. A software implementation, the Joint Scalable Video Model, JSVM, is available for download and experimentation [iii].

H.264 SVC supports three main types or classes of scalability (Figure 10.3):

1. **Temporal scalability**: The base layer is coded at a low temporal resolution or low frame rate; adding enhancement layers increases the frame rate of the decoded sequence.
2. **Spatial scalability**: The base layer is coded at a low spatial resolution; adding enhancement layers increases the spatial resolution of the decoded sequence.
3. **Quality scalability**: The base layer is coded at a low visual quality using a high QP; adding enhancement layers increases the visual quality of the decoded sequence.

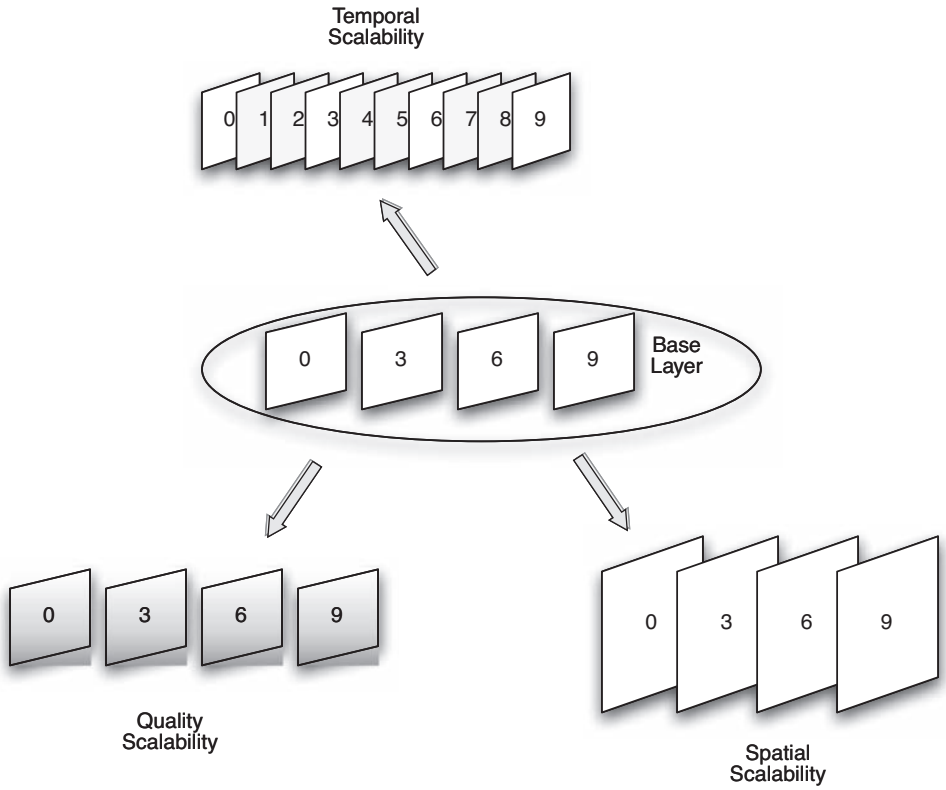


Figure 10.3 Overview of scalability types

Example 1 – Quality Scalability:

Layer 0, base layer, is coded at a bit rate of 200kbps, at CIF resolution, 352 × 288 luma samples per frame, and at 30 frames per second. Enhancement layer 1 is coded at a bit rate of 520 kbps using quality scalability, i.e. the frame rate and resolution stay the same. A low-quality sequence is obtained by sending the base layer over a 200kbps channel and decoding it. A high-quality sequence is obtained by sending base + enhancement layers over a 720kbps channel. (See Table 10.1).

Table 10.1 Example: Quality Scalability

	Encoder		Decoder	
	Base	Enhancement	Base	Base + Enhancement
Resolution	352 × 288	352 × 288	352 × 288	352 × 288
Frames per second	30	30	30	30
Bitrate	200kbps	520kbps	200kbps	720kbps

Example 2 – Spatial and Temporal Scalability:

Base Layer 0 is coded at a bit rate of 400kbps, with spatial resolution 320 × 240 luma samples and at 15 frames per second. Enhancement Layer 1 is coded at a bit rate of 800kbps, a spatial resolution of 640 × 480 samples and at 30 frames per second. In this example, spatial and temporal scalability are used simultaneously. Decoding Layer 0 only (400kbps) gives a low-resolution, low frame rate sequence; decoding layers 0 and 1 (a total of 1200kbps) gives a higher resolution, higher frame rate output sequence. See Table 10.2.

Table 10.2 Example: Spatial + Temporal Scalability

	Encoder		Decoder	
	Base	Enhancement	Base	Base + Enhancement
Resolution	320 × 240	640 × 480	320 × 240	640 × 480
Frames per second	15	30	15	30
Bitrate	400kbps	800kbps	400kbps	1200kbps

10.2.5 Temporal scalability

In a sequence coded with temporal scalability, the base layer 0 is coded at the lowest temporal resolution, i.e. the lowest frame rate. Successive enhancement layer(s), when decoded with the base layer, provide progressively higher decoded frame rates. Figure 10.4 shows a sequence encoded as three temporally scalable layers. Layer 0 is coded at a frame rate F_0 and consists of coded frames 0, 6, 12, etc. An H.264/AVC decoder can decode layer 0 in isolation.

Layer 1 consists of frames 3, 9, 15, . . . etc. A decoder may decode layers 0 and 1 to produce a higher-rate sequence (Figure 10.5) at $2F_0$ frames per second. Layer 2 consists of frames 1, 2, 4, 5, 7, 8, . . . etc and a decoder that decodes layers 0, 1 and 2 can deliver an output sequence (Figure 10.5) at $6F_0$ frames per second.

Temporal scalability can be achieved using the P- and/or B-slice coding tools available in H.264/AVC. The examples in Figure 10.4 and Figure 10.5 are developed using the 'Hierarchical' or 'Pyramid' Group of Pictures structure discussed in Chapter 6 (section 6.4.7). For

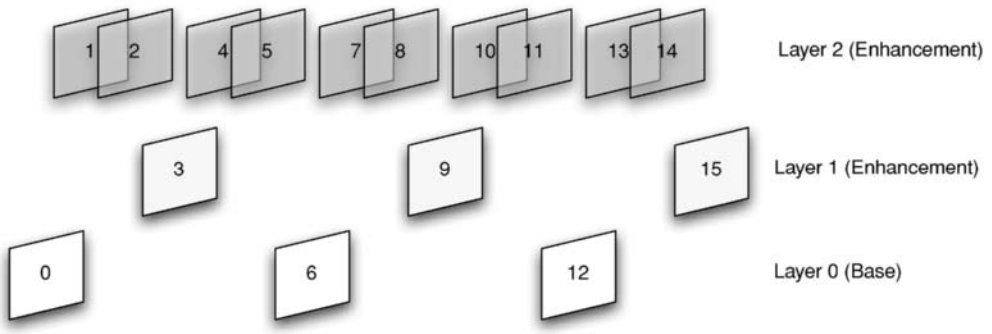


Figure 10.4 Temporally scalable sequence, 3 layers

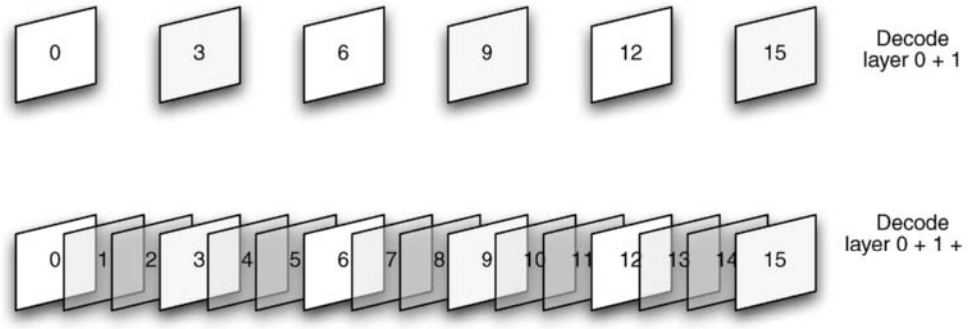


Figure 10.5 Decoding a temporally scalable sequence

completeness, Figure 10.6 shows the prediction directions for this hierarchical structure. The base layer consists of the I-slices every 12 slices and the first set of B-slices, B6, etc. Layer 1 consists of the second set of B-slices (3, 9, 15, etc) which are predicted from the base layer. Layer 2 consists of the remaining B-slices which are predicted from layers 0 and 1. Hence the following sub-sets may be independently decoded:

Layer 0, I0, B6, I12, etc

Layer 0 + Layer 1, I0, B3, B6, B9, etc.

Layer 0 + Layer 1 + Layer 2, I0, B1, B2, B3, etc.

Because the necessary prediction tools, i.e. P or B pictures used for reference, are supported in the Main and High Profiles of H.264/AVC, temporal scalability can be achieved **without** the need for any extensions to the core H.264/AVC standard.

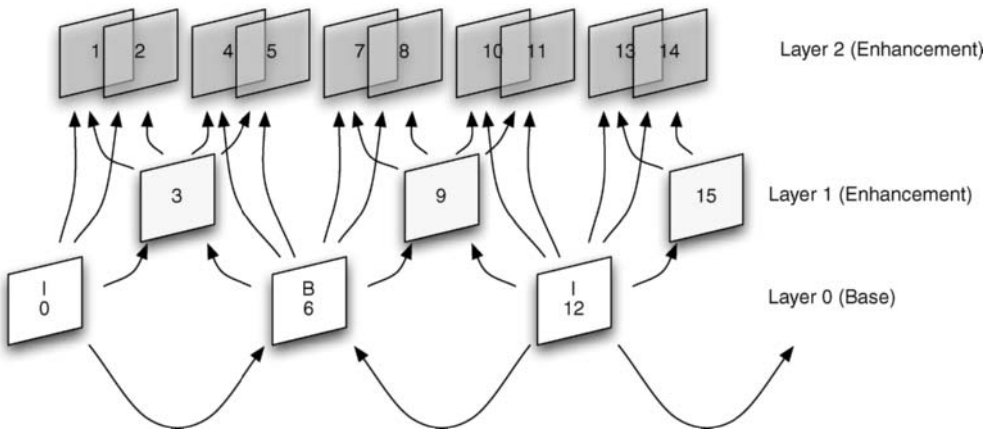


Figure 10.6 Hierarchical prediction structure

10.2.6 Quality scalability: overview

The base layer is coded using a particular quantizer parameter QP to produce the Layer 0 bitstream (Figure 10.7). Consider a single video frame A. At the encoder, this coded frame is decoded from the base layer and reconstructed (frame A'0). Frame A is re-coded at a lower QP and hence a higher quality, with decoded frame A'0 available as a prediction reference, to produce the enhancement layer bitstream Layer 1. Note that A'0 will typically be a very effective prediction reference as it is identical to frame A except for distortion introduced by compression. Note that the 'usual' prediction sources, previously-coded frames in the decoded picture buffer, are also available for prediction of each macroblock.

A base layer decoder simply decodes frame A0. An enhancement layer decoder requires the decoded A, a prediction reference, to reconstruct the higher-quality frame A1.

This process may be repeated to form a 'cascade' of layers 0, 1, 2, etc, each layer (a) using the reconstructed frame from the layer below as a prediction reference and (b) using a progressively lower QP. Note that SVC provides tools that make it possible to reconstruct the enhancement layers without fully decoding the base layer information (constrained inter-layer prediction).

10.2.7 Spatial scalability: overview

In the case of Spatial Scalability (Figure 10.8), the base layer has the lowest resolution and successive enhancement layers can be decoded to provide higher resolution decoded frames or fields.

An input video frame A is downsampled at the encoder to produce a low-resolution version A'. Frame A' is coded to produce the base Layer 0 and can be decoded to give low-resolution output frame A0. The encoder reconstructs A0 and upsamples it to produce a reference frame that has the same effective resolution as the original (A). This reference frame is used as a prediction reference, enabling the encoder to produce the enhancement Layer 1. The up-sampled A0 will typically be a good prediction reference for frame A because it is the same frame, with distortions due to downsampling, coding and up-sampling.

An enhancement layer decoder upsamples A0 and uses it to reconstruct the decoded enhancement frame A1. As with Quality scalability, this process may be repeated to give a cascade of layers 0, 1, 2, etc. The highest-resolution layer has the same resolution as the original sequence; lower layers are coded at progressively smaller resolutions.

10.2.8 Spatial scalability in detail

As discussed above, the base layer of a spatially scalable bitstream is encoded using the usual H.264/AVC tools described in earlier chapters. Coding a macroblock in an enhancement layer requires a number of changes depending on the type of prediction from the lower layer. H.264 SVC goes beyond the basic approach of up-sampling the lower layer (section 10.2.7) and provides several new prediction modes that improve the coding performance of spatially scalable compression.

In an enhancement layer coder, e.g. Encoder 1 in Figure 10.8, there are a number of options for predicting the current macroblock. First, all of the usual prediction options are available: intra modes using samples from the current frame at the enhancement layer resolution, inter

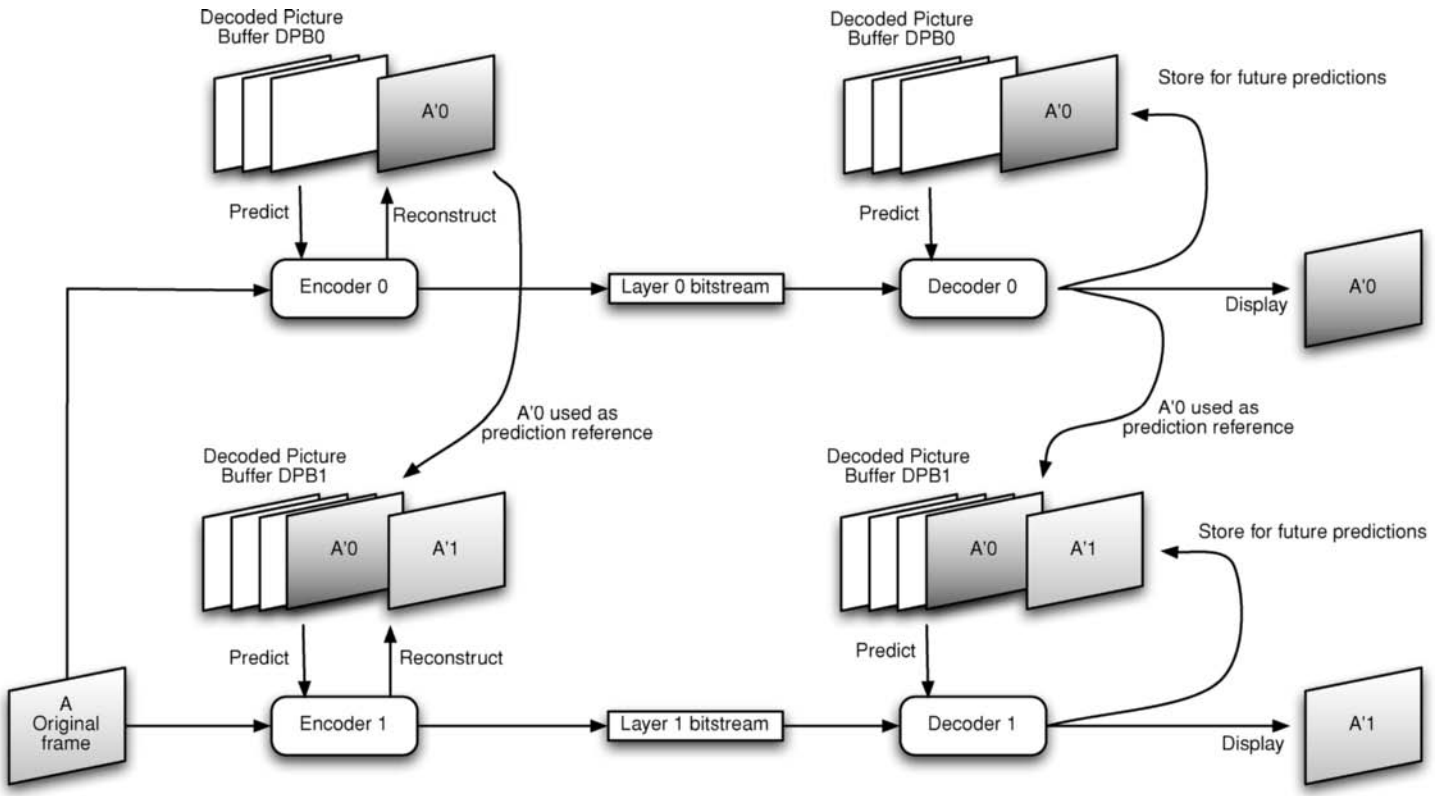


Figure 10.7 Quality Scalability

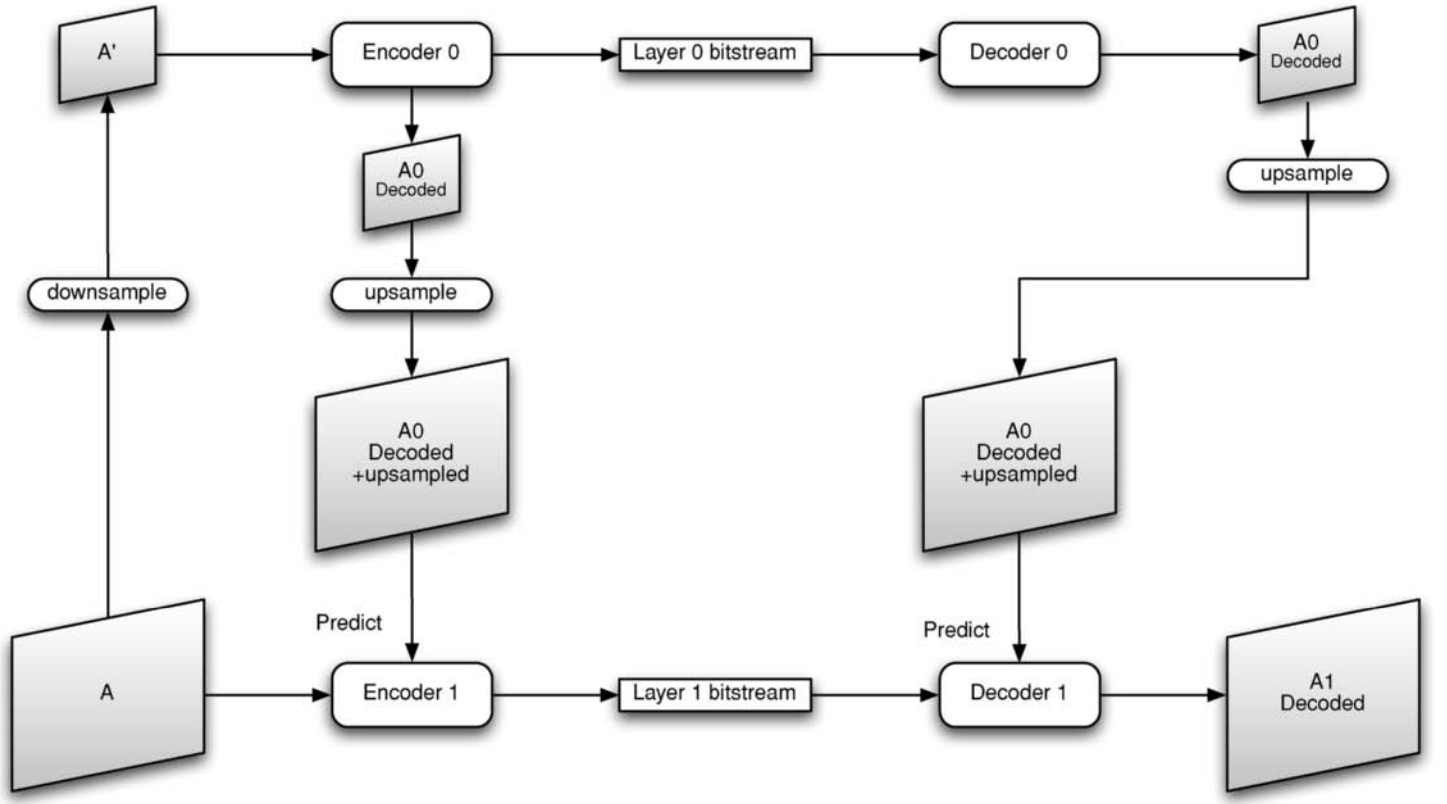


Figure 10.8 Spatial scalability, two layers

modes using samples from previously coded and reconstructed frames at the enhancement layer resolution. Second, the following further options are available, using the upsampled lower layer, the base layer in Figure 10.8, or the next lower-resolution enhancement layer, as a **reference layer**. Note that the current MB position corresponds to a smaller block in the lower-resolution layer. The following discussion assumes an 8×8 corresponding block in the lower, reference layer, so-called dyadic scaling, or $2 \times$ horizontal and vertical resolution in the enhancement layer. However, arbitrary inter-layer scaling factors are supported by H.264/SVC.

Prediction options:

1. Upscale the reference layer.
   For Intra blocks, scale the reference layer to the same resolution as the current layer (Figure 10.8) and use the reference layer as an extra prediction reference.
2. Base Mode: Use the prediction choices from the reference layer block.
   When a Base Mode Flag is set to 1, only a residual is sent in the enhancement layer, with no extra prediction choices, i.e. no intra prediction modes or inter partitions, references and motion vectors.

   If the co-located block in the reference layer was coded in Intra mode, the reconstructed intra block from the reference layer is up-sampled using a 4-tap Finite Impulse Response filter to produce a prediction for the current MB. This prediction is subtracted from the MB to produce the enhancement layer residual.

   If the co-located block in the reference layer was coded in Inter mode, the enhancement layer block is predicted using Inter prediction, with (a) the same reference picture indices, (b) partition choices that are up-sampled from the partitions in the reference layer and (c) motion vectors scaled up from the reference layer motion vectors.
3. Motion vector prediction from the reference layer.
   If a Motion Prediction Flag is set to 1, the current enhancement layer macroblock partition is predicted using Inter prediction with (a) the same reference picture indices as the corresponding reference layer block and (b) motion vector differences (MVD) created using the up-scaled motion vectors of the reference layer as predictors (Chapter 6).
4. Residual prediction.
   When a Residual Prediction Flag is set to 1, the enhancement layer residual is predicted from the reference layer residual. First, the reference layer residual is up-sampled using bi-linear interpolation and this up-sampled residual is subtracted from the original enhancement layer block. Then, the enhancement layer residual is formed using any of the methods described above, i.e. conventional intra/inter prediction or base mode prediction. The resulting difference signal is transformed, coded and transmitted as usual (Chapter 7).

Note that inter-layer prediction is constrained as follows. First, the only enhancement layer macroblocks which may be coded with inter-layer intra prediction are those for which the co-located reference samples are intra coded (constrained intra prediction). Second, constrained intra prediction is mandatory for inter-layer prediction of higher layers. This means that intra coded macroblocks in reference layers can be constructed without having to reconstruct any inter coded macroblocks. Hence each layer may be decoded using a single motion compensation loop (single loop decoding), resulting in significantly lower decoder complexity than scalable video coding in earlier standards [i].

10.2.9 Quality scalability in detail

H.264/SVC supports Coarse Grain and Medium Grain quality scalability. Coarse Grain Quality Scalability (CGS) is effectively a special case of Spatial Scalability in which the upsampling / downsampling factor is 1. This means that the enhancement layer resolution is the same as the reference layer resolution (Figure 10.7). The enhancement layer is coded at a lower QP and hence a higher quality than the lower, reference layer. All the 'spatial' scalable coding tools described above may be use to predict the enhancement layer frame from the reference layer reconstructed frame.

A typical application of quality scalability is to provide versions of the sequence coded at different bitrates and quality levels, so that, for example, lower bitrate sub-sequences may be extracted for transmission over channels with different capacities. With CGS, the number of sub-sequence bitrates is limited by the number of layers. Providing a large number of bitrate options using CGS requires a large number of layers, which tends to be complex and inefficient to code.

Medium Grain Quality Scalability (MGS) addresses this limitation and makes it possible to extract substreams at a wide range of bitrates from a scalable bitstream with a small number of quality layers. Using MGS, any NAL unit in an enhancement layer may be discarded to leave a fully decodeable bitstream. This makes it possible to produce a variety of output bitrates. For example, discarding an arbitrary number of enhancement layer NAL units makes it possible to meet an arbitrary bit rate target, within a certain margin of error. Figure 10.9 shows an example. The complete scalable bitstream consists of Base Layer NALUs and enhancement

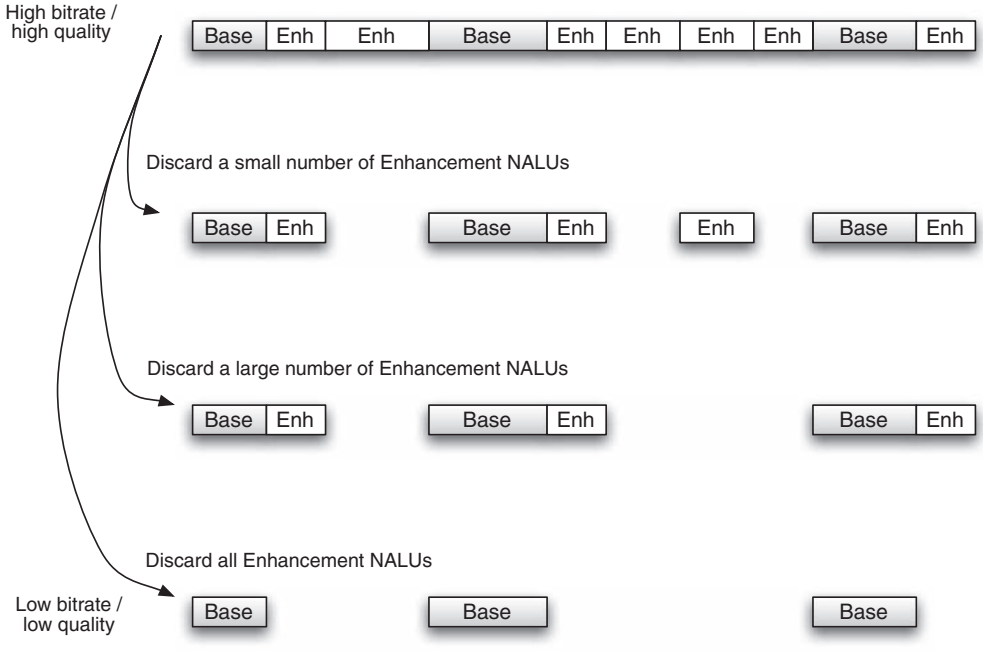


Figure 10.9 Medium Grain Quality Scalability

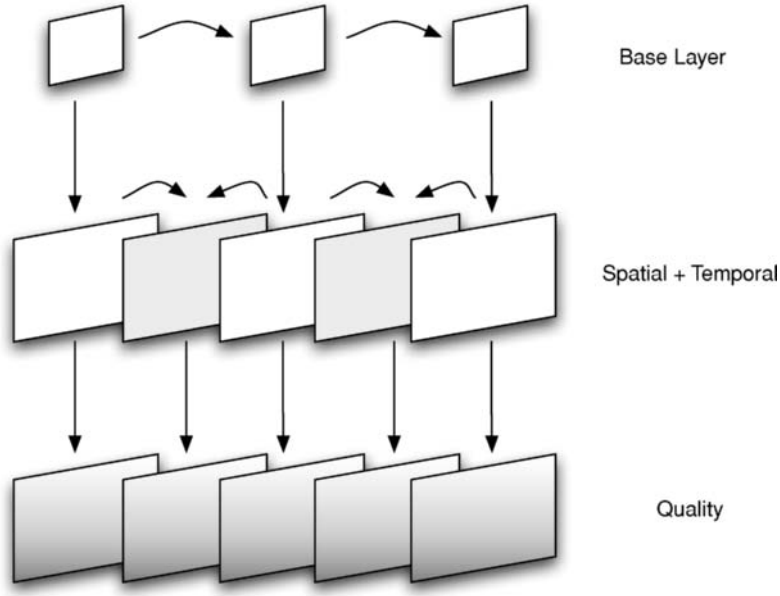


Figure 10.10 Combined Spatial, Temporal and Quality scalability

layer NALUs as shown. Using MGS, selected Enhancement Layer NALUs may be discarded to provide sub-streams at a progressively lower bitrate. The lowest bitrate / quality point is provided by the Base Layer stream, which consists only of Base Layer NALUs. SVC specifies that motion compensated prediction parameters must not change between Base and Enhancement layer representations of certain so-called Key Pictures. This prevents "drift" between the motion compensated reconstruction of these Key Pictures at the encoder and decoder. Drift is therefore restricted to non-Key Pictures.

10.2.10 Combined scalability

H.264/SVC provides considerable flexibility in the construction of a scalable bitstream, making it possible to mix Spatial, Temporal and Quality scalability. For example, in Figure 10.10, the base layer is up-sampled spatially and the up-sampled frames are used as references for B-slices to produce a layer with spatial and temporal scalability. This is then used to predict a further layer at the same spatio-temporal resolution but at a higher bitrate and quality.

10.2.11 SVC performance

For a given sequence that is required to be delivered at a range of bitrates, a key performance question is as follows – does scalable coding give a **smaller** or **larger** bitrate than simulcast coding?

Example:

Two coded versions of a sequence are required to be delivered across a network. The choices are (a) code low quality and high quality versions independently using AVC and deliver each sequence (simulcast) or (b) code a low quality Base Layer and an Enhancement Layer that can be decoded with the Base Layer to give the high quality version using scalable decoding. Figure 10.11 illustrates the total bitrates of the two choices. In this case, the base layer of the scalable version is coded at a **higher** bitrate than the corresponding simulcast version. This is generally advisable in order to provide a good reference for predictions of the enhancement layer. However, the total bitrate of the scalable version is **lower** than the combined simulcast bitrate. In this example, scalable coding is more efficient in terms of total bitrate.

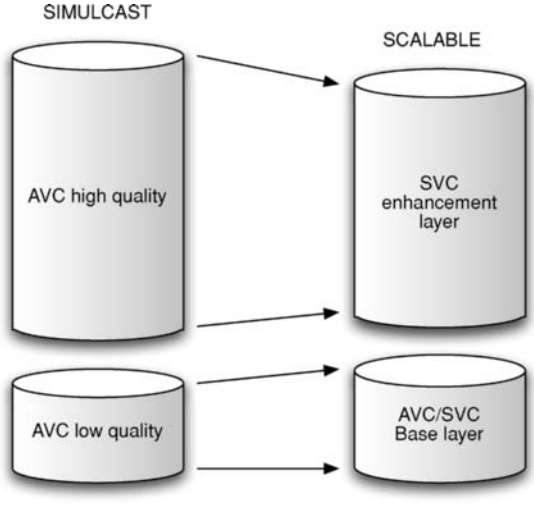


Figure 10.11 Simulcast vs. scalable bitrates

MPEG Technical Report N9577 [iv] compares the performance of SVC and AVC. This report describes a series of tests in which the same video clips were coded using (a) AVC and (b) SVC. The clips were coded at a number of quality levels and resolutions. In each case, a low-rate clip and a high-rate clip were produced. The low-rate clip was coded (a) using AVC and (b) as the base layer of a scalable stream. The high-rate clip was coded (a) as a separate AVC stream and (b) as the enhancement layer of a scalable stream. The operating bitrates were chosen such that the visual quality of the decoded sequences were approximately the same at each rate point, low and high. The perceived quality of each clip was measured by combining the opinion scores of a number of viewers (Mean Opinion Score, see Chapter 2).

Selected results from [iv] are shown in Figure 10.12, Figure 10.13 and Figure 10.14. Quality scalability with two layers is tested in Figure 10.12 for the sequences 'Ofer' and 'Paris'. The lower rate point represents a 30fps CIF sequence coded at a low bitrate. The SVC base layer rate and subjective quality, denoted as 'Quality of the Highest Layer', is slightly higher than the AVC low rate stream. The upper rate point represents the **total** bitrate of (a) both the AVC simulcast streams or (b) both the SVC layers. Note that the x-axis is **cumulative** bitrate. It is clear that in this case, SVC achieves the same quality as AVC with a lower combined bitrate.

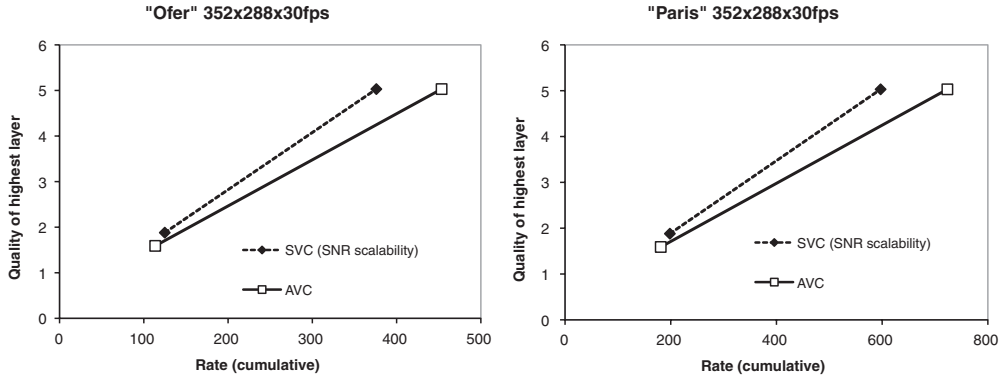


Figure 10.12 Quality scalability at CIF resolution

As illustrated in Figure 10.11, the total rate for two simulcast streams is higher than the total rate for two SVC layers.

Figure 10.13 shows similar results for combined spatial and temporal scalability. The lower rate point represents 15fps, 320 × 240 video and the upper rate point represents 30fps, 640 × 480 video, coded as (a) two simulcast streams or (b) base and enhancement SVC layers. Once again, the combined rate of the SVC layers is lower than the combined rate of the AVC streams, for the same or better visual quality. It is interesting to note that 'Crew' SVC base layer is actually given a very high quality score by the viewers. This may be due to the fact that the low-resolution sequence (320 × 240 × 15fps) has fewer distortion artefacts than the high-resolution sequence (640 × 480 × 30fps) and so is ranked approximately the same.

Finally, Figure 10.14 shows the results for the high-definition sequence 'AlohaWave' with spatial scalability or AVC simulcast delivering 720p / 50fps and 1080p / 50fps streams. Once again, SVC delivers the same or better quality at a lower combined rate than AVC. Note that subjective quality appears to **drop** as the rate and resolution increase. Again, this is probably due to the fact that more obvious coding distortion appears in the higher-resolution sequence.

The results presented in [iv] show a clear, if relatively modest, benefit from using SVC to deliver multiple versions of a sequence at different rate, quality and resolution points.

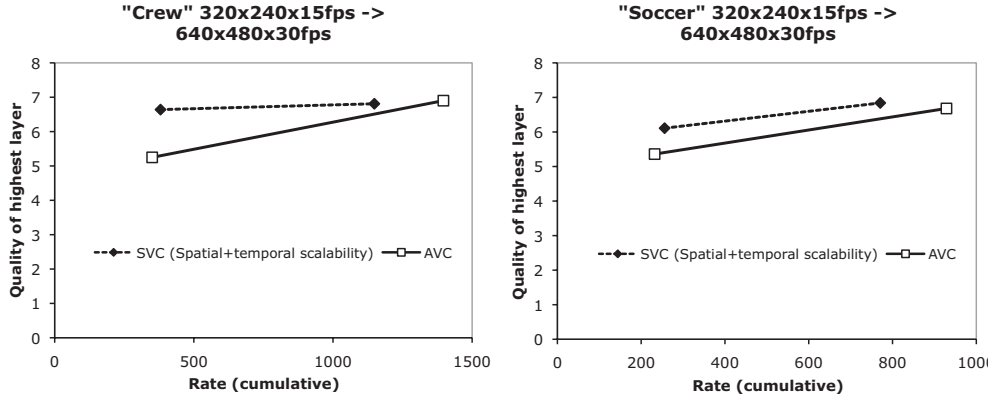


Figure 10.13 Spatial + temporal scalability, CIF → 4CIF resolution

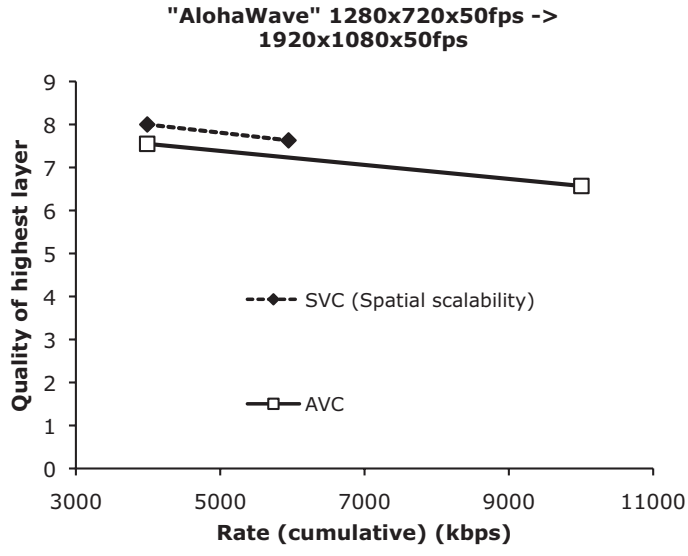


Figure 10.14 Spatial scalability, 720p → 1080p resolution

Despite the fact that scalable coding has been an active research topic since the early 1990s and has been incorporated into earlier standards such as MPEG-2 Video and MPEG-4 Visual, it has not been widely adopted by the video coding industry, perhaps because the cost and complexity outweighs the technical advantages. However, there are some indications that H.264 SVC is a more attractive commercial proposition, with recent product announcements in the videoconferencing market [v,vi].

10.3 Multiview Video Coding

Multiview video is video data that incorporates multiple concurrent versions of a particular scene. An example is shown in Figure 10.15; these three images are three simultaneous views of the same real-world scene, taken from different viewpoints. Potential implementations of multiview video include:

(a) Stereoscopic video. A stereo pair of views of the scene are combined, e.g. using data glasses or autostereoscopic displays, giving the illusion of a three-dimensional view, albeit with a limited viewing angle (Figure 10.16).
(b) 3D video. Multiple actual or rendered views of the scene are presented to the viewer e.g. using 'virtual reality' glasses or an advanced autostereoscopic display, so that the view

Figure 10.15 Three views of the same scene

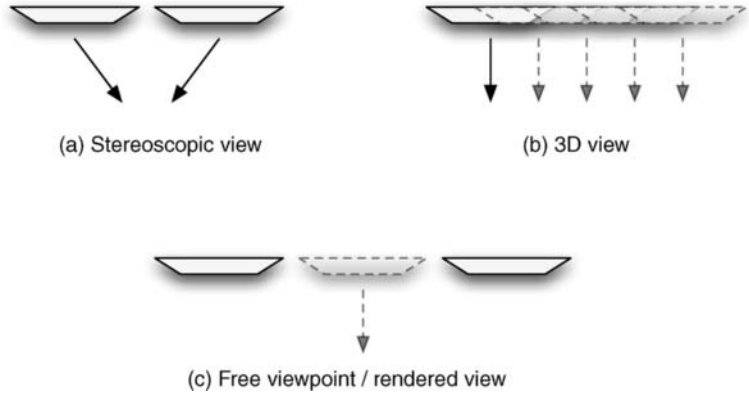


Figure 10.16 Multiview video : view examples

changes with head movements and the viewer has the feeling of 'immersion' in the 3D scene (Figure 10.16).

(c) Free-viewpoint video. A limited number of views of the scene are available, e.g. from multiple cameras at a sports game or multiple surveillance cameras. The viewer may select an arbitrary viewing angle. If this view does not exist, it is rendered or created from the available 'real' views (Figure 10.16).

Applications of multiview video include 3D television, advanced surveillance systems, immersive teleconferencing and gaming.

In a similar way to scalable video, multiview video content has inherent redundancy and the purpose of Multiview Video Coding (MVC) is to capitalise on this redundancy and efficiently code a multiview video scene. Figure 10.17 shows sequences of video frames corresponding

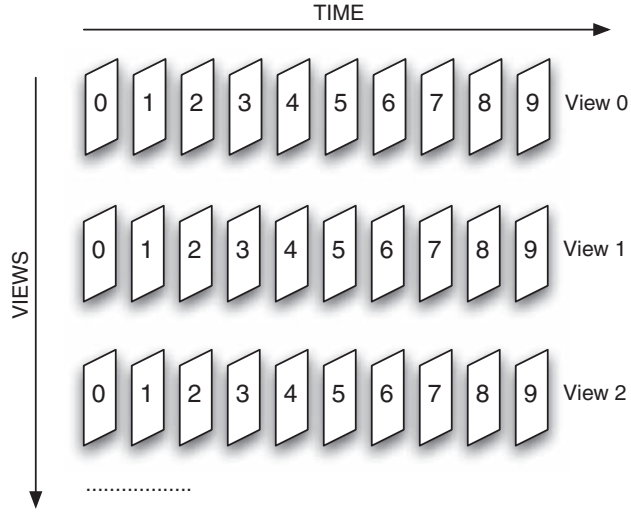


Figure 10.17 Multiview video : views and frames

to a series of views of a multiview scene. Each view consists of a series of frames or fields that may be coded as a separate H.264/AVC stream, i.e. simulcast coding of each view. However, there is likely to be a correlation between views, particularly if the camera positions are close together. Hence frame 0 of view 0 may be strongly correlated with frame 0 of view 1, if the camera positions are close; frame 0 of view 1 may be correlated with frame 0 of view 2; and so on.

10.3.1 H.264 Multiview Video Coding

The inherent redundancies in a multiview scene can be exploited by introducing predictions between views, i.e. inter-view prediction structures. This has required an extension to H.264/AVC, known as H.264 Multiview Video Coding (H.264 MVC). H.264 Multiview Video Coding is incorporated as Annex H into a draft revision of H.264/AVC [vii, viii]. Reference software is available at [ix].

An example of interview prediction is shown in Figure 10.18. View 0 (top) is predicted using a hierarchical GOP structure (Chapter 6) using conventional H.264/AVC tools. Each GOP consists of an I slice or 'key picture' followed by seven B slices. This means that View 0 can be decoded by an AVC or MVC decoder and can be considered as the Base layer or Base view. Each of the other views uses a similar prediction structure, except that the key pictures are now P slices, predicted from an I or P slice in the previous view. Inter-view correlation

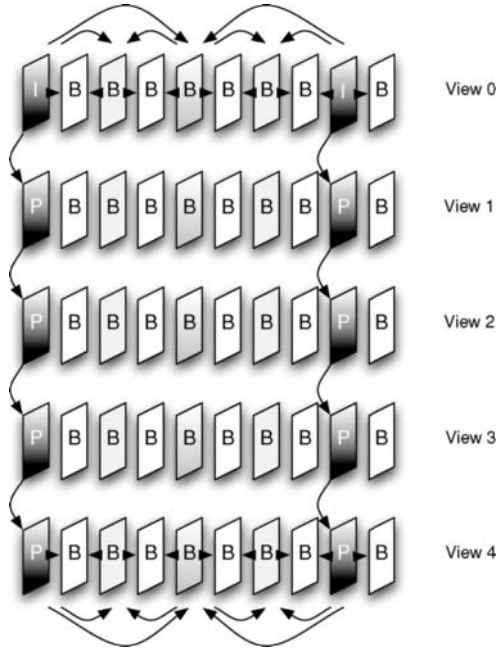


Figure 10.18 Interview prediction of key frames

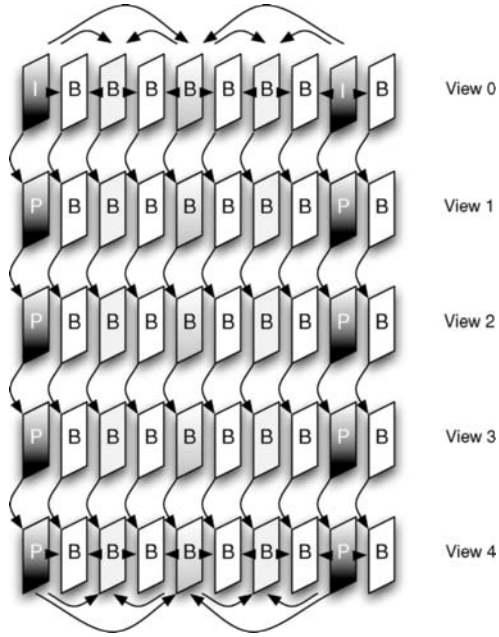


Figure 10.19 Inter-view prediction of all frames

means that the P slices in views 1, 2, 3 . . . are likely to be more efficiently coded than I slices in the same positions.

A more complex prediction structure includes interview prediction for every picture (Figure 10.19). Hence any B- or P-slice in views 1, 2, 3, . . . has a picture in another view available as a reference for prediction.

Annex H to H.264/AVC specifies a number of additions to the basic H.264 syntax to support MVC, including:

- Sequence Parameter Set: specify views and anchor or key picture references.
- Reference Picture List: structured to include support for inter-view prediction.
- NAL Unit order: modified to allow the use of a Prefix NALU, containing extra information about the Base view. This special Prefix NAL Unit may be discarded by an AVC decoder that is not MVC-compatible, so that the base view may still be decoded.
- Picture numbering and reference indices: modified to support multiple views.

MVC has to face the same challenge as SVC, namely, are the benefits of MVC worth the extra complexity compared with conventional simulcast coding of multiple views? Multiview video applications are still at a relatively early stage and there is not yet evidence that H.264 MVC will become popular in the industry as a coding technology. However, the popularity of stereoscopic ("3D") films such as "Avatar" is leading manufacturers to develop stereoscopic TV and Blu-Ray solutions.

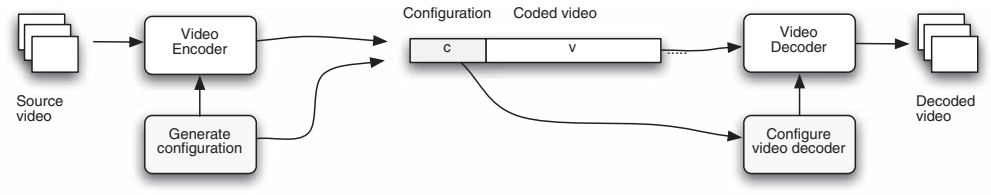


Figure 10.20 Overview of configurable video codec

10.4 Configurable Video Coding

H.264/AVC is a relatively recent addition to a series of popular video coding formats, starting with the early standards ITU-T Recommendation H.261 and ISO/IEC MPEG-1 and including an increasing number of standard and non-standard formats. Coding formats such as MPEG-2 Video, MPEG-4 Visual, VC-1 and H.264/AVC are widely used in current devices and systems. None of these formats or standards is inter-operable, i.e. video coded in one format can only be decoded by a decoder supporting the same format. As the digital video market continues to grow, this presents an increasing problem for device and codec manufacturers as they are faced with the need to support multiple, incompatible coding standards [x] leading to over-designed decoding products. The challenge of efficiently supporting multiple coding formats has led to increased interest in configurable video coding solutions. All of the current formats have certain aspects in common, such as block-based operation, inter-frame prediction and various forms of transform and entropy coding. Finding a way to exploit these commonalities may make it possible to efficiently and flexibly support multiple video coding formats.

The general concept of a configurable video codec is illustrated in Figure 10.20. At the encoder side, video is compressed using a particular configuration of video encoder, e.g. using a standard or non-standard coding format. Configuration information and coded video data are sent to the decoder. The configuration information is used to generate or configure a video decoding algorithm. The configured video decoder then proceeds to decompress the coded video sequence.

A configurable coding system has the potential to greatly increase the flexibility of a video codec, making it possible to re-configure the codec to handle multiple existing video formats, or 'upgrading' the decoder to handle a new coding format. Key questions for the design of such a system include:

- Should the configuration be completely flexible, or should it be limited to a number of pre-defined options?
- When should configuration occur – at the start of a communication session, during a communication session or both?
- Can a re-configurable codec achieve the same computational performance as a 'hard-wired' or fixed software or hardware codec?
- What impact does this approach have on issues such as inter-operability with existing codecs, random access to video streams and intellectual property questions such as license rights to coding algorithms?

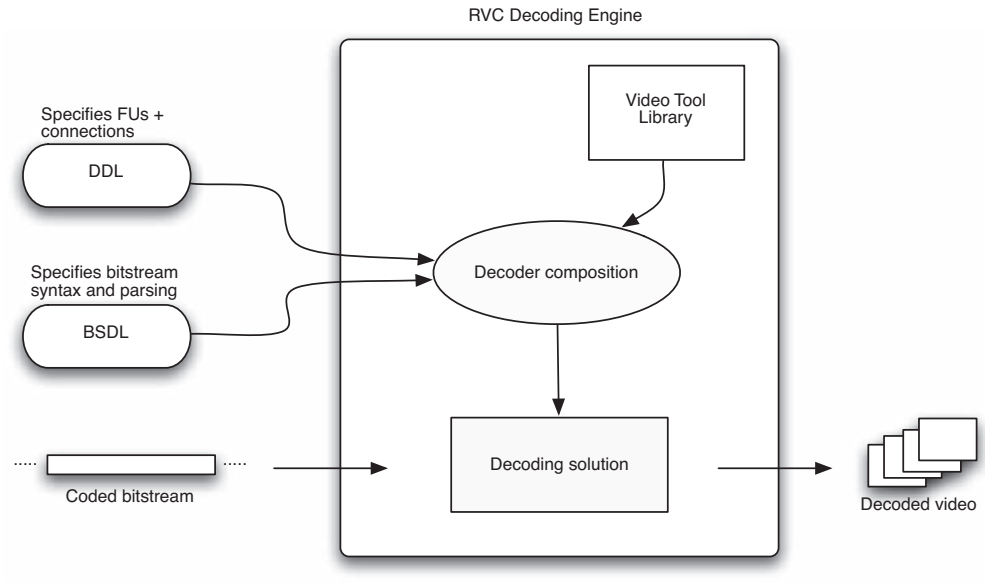


Figure 10.21 Reconfigurable Video Coding scenario

10.4.1 MPEG Reconfigurable Video Coding

MPEG's Reconfigurable Video Coding (RVC) initiative builds on concepts originally proposed over ten years ago [xi] and aims 'to provide a framework allowing a dynamic development, implementation and adoption of standardized video coding solutions with features of higher flexibility and reusability' [xii].

The RVC sub-group of MPEG has developed two standards that enable flexible reconfiguration of video codecs to support a number of coding formats in an efficient way. In the RVC model, a decoder is specified as an interconnected set of Functional Units (FUs) which are decoding tools such as inverse transforms and entropy decoders. The available FUs are specified in the Video Tool Library (VTL) standard [xiii]. A particular decoder is defined by a Codec Configuration Representation [xiv], which describes a bit stream format and a set of FU interconnections and parameters. The Codec Configuration Representation is specified prior to starting a decoding session. Hence, an RVC video decoder is constructed from a set of pre-defined decoding tools. An existing or new video format can be supported by reconfiguring the RVC decoder, provided the format uses standard FUs from the Video Tool Library.

Figure 10.21 illustrates a typical RVC decoding scenario. In order to decode a video bitstream, the decoder needs to know (a) how to parse the bitstream and extract the coded data elements and (b) how to decode these elements. The RVC decoding engine receives BSDL and DDL specifications in compressed form. The decoder composition module generates a decoding solution, an actual video decoder, based on the BSDL and DDL specifications. It makes use of selected FUs from the Video Tool Library and connects these according to the DDL. Once the decoding solution has been generated, it can then decode the video bitstream.

The MPEG RVC approach has a number of potential benefits. A decoder can be modified to decode a different format by sending new BSDL/DDL descriptions, enabling efficient support for multiple coding formats. A non-standard coding format can be supported provided it uses FUs available to the decoder, i.e. FUs in the decoder's VTL. In practical terms this means that a new format should use FUs standardized by MPEG. A new coding tool can be proposed to MPEG for standardization as a new FU – a potentially faster, simpler route than creating a complete new video coding standard.

Two key constraints deliberately selected for the RVC model are:

1. A coding format must use combinations of coding tools (FUs) that are specified in the VTL. Introducing a new tool that is not in the VTL is likely to require a lengthy process of standardizing and disseminating a new version of the VTL.
2. The decoder configuration is fixed for the duration of a communication session, i.e. there is limited potential to change the coding algorithm if and when the characteristics of a video sequence change.

10.4.2 Fully Configurable Video Coding

Fully Configurable Video Coding (FCVC) has been proposed as an alternative to, or an evolution of, the RVC approach. FCVC differs from RVC in two ways:

1. The transmitted configuration information completely describes the video decoding algorithm. This means that (a) a library of pre-defined video coding tools is not necessary at the decoder and (b) any existing or new video coding algorithm can be configured.
2. Re-configuration can occur at any point during a video communication session. This makes it possible to adapt the video coding algorithm dynamically, for example changing aspects of the algorithm based on the statistics of the video sequence.

In the FCVC framework (Figure 10.22) a common decoding engine, the Universal Video Decoder (UVD), can be configured to decode any video sequence or syntax [xv]. In a typical application, the UVD initially has limited or no knowledge of the decoding methods required for a particular video bitstream. The encoder sends a set of configuration commands, the Decoder Description Syntax (DDS) which define the decoder design in terms of a set of primitive operations and interconnections. The UVD generates and connects new functional processing units according to these commands and can then proceed to decode the video bitstream. At a later point, the encoder may signal a change in configuration by sending new Decoder Description Syntax; the UVD implements the change and continues to decode the bitstream using the changed syntax or functionality. FCVC has the following benefits compared with conventional approaches:

(i) Any video decoding processes may be defined and created using a set of low-level configuration commands, or primitives for describing decoding operations. New algorithms can be implemented in decoders in a very short timescale which implies a short time-to-market.

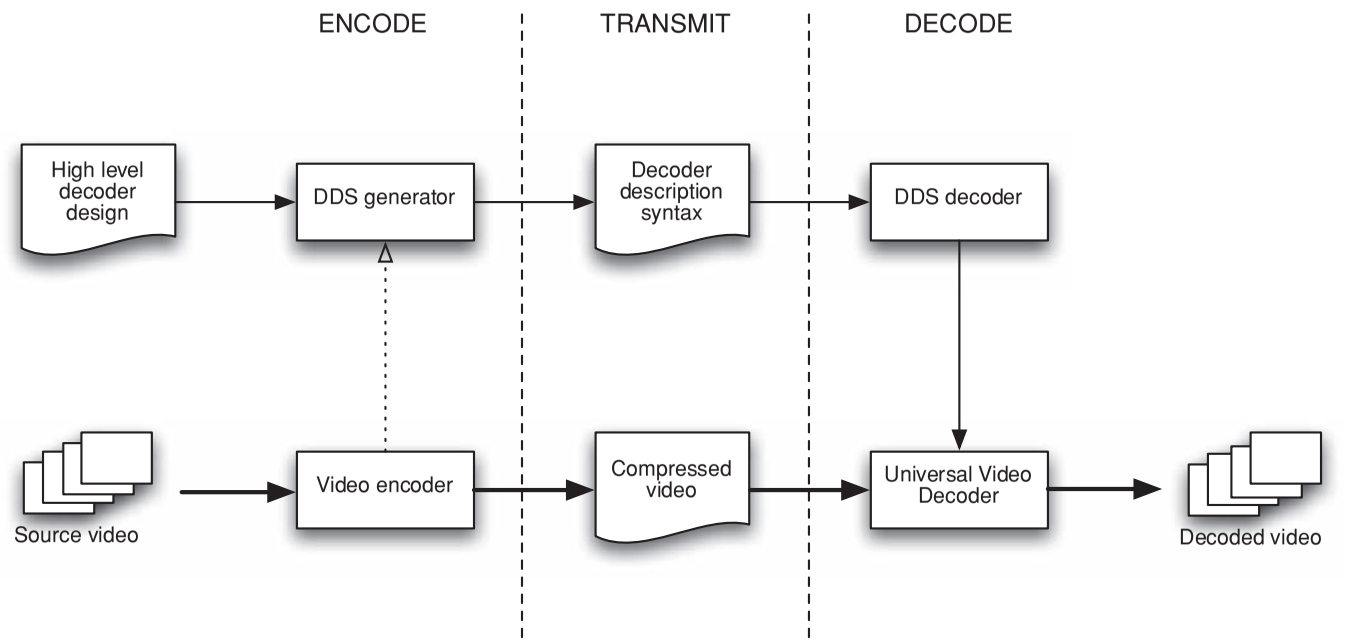


Figure 10.22 Fully Configurable Video Coding framework

(ii) Re-configuration can be carried out dynamically, enabling on-the-fly adaptation. This allows the codec to change its configuration to suit the video sequence, delivering optimal or near-optimal compression efficiency.
(iii) A single Universal Video Decoder can be re-configured to support any existing or new coded video format.
(iv) The Universal Video Decoder is programmed with only the tools it needs.

Recent efforts have focussed on developing and optimizing a prototype FCVC system [xvi] and on investigating the potential for combining aspects of FCVC and RVC [xvii].

10.5 Beyond H.264/AVC

First standardized in 2003, H.264/AVC is now a relatively mature technology. Other formats such as VC-1 and AVS can arguably offer similar performance but H.264 is certainly one of the leading formats in terms of compression efficiency at the time of writing (early 2010).

The Moving Picture Experts Group (MPEG) and Video Coding Experts Group (VCEG) are examining the need for a new video compression standard. Following a Call for Evidence [xviii] several proposals for improved video compression were presented at an MPEG meeting in July 2009. The consensus was that (a) there is likely to be a need for a new compression format, as consumers demand higher-quality video and as processing capacity improves and (b) there is potential to deliver better performance than the current state-of-the art. A number of different techniques were proposed, including decoder-side motion estimation, larger macroblock sizes (up to 32×32), more sophisticated in-loop deblocking filters, adaptive transform sizes and improved intra prediction. In general, all of these proposed algorithms offer the potential for better compression performance at the expense of increased computational complexity.

Results of subjective comparison tests of the new proposals led the committee to conclude that 'for a considerable number of test cases significant gain over AVC High Profile could be achieved' [xix]. For many of the tested video sequences, the proposed new algorithms led to improvements in subjective quality of one point or more on the MOS scale (Chapter 2). This implies that there is scope for a new coding format that significantly out-performs H.264/AVC.

The current plan is to set up a Joint Collaborative Team (JCT) of MPEG and VCEG representatives to work on a new video coding standard. Proposals for the new standard will be reviewed in 2010 and a new standard could be finalized around 2012/2013. It will aim to deliver significantly better compression performance than H.264/AVC, probably at a higher computational cost. A working title for the new standard is still under discussion.

10.6 Summary

H.264/AVC was always conceived as a toolkit of coding algorithms that could be extended to meet future needs. Recent extensions have included the Scalable Video Coding (SVC) and Multiview Video Coding (MVC) Annexes, which build on the basic standard to add support for efficient coding of multiple, correlated video streams. SVC is designed to accommodate multiple versions of the same video scene, at different spatial and temporal resolutions and at different bitrates, whereas MVC is intended to handle multiple closely-related views of a scene. For both SVC and MVC, better compression efficiency is achieved at the expense of

increased complexity. Despite some market interest in SVC, it remains to be seen whether either of these extensions will be widely adopted.

Looking beyond H.264/AVC, two trends considered in this chapter are to (a) 'open up' video coding by allowing flexible re-configuration of coding tools and to (b) develop a new, higher-performance standard that improves compression efficiency. These trends are not necessarily mutually exclusive. There is likely to be a continued need for better compression efficiency, as video content becomes increasingly ubiquitous and places unprecedented pressure on restricted network connections. At the same time, the challenge of handling ever more diverse content coded in a wide variety of formats makes reconfigurable coding a potentially useful prospect.

10.7 References

i. H. Schwarz, D. Marpe and T. Wiegand, 'Overview of the scalable video coding extension of the H.264/AVC standard', IEEE Transactions on Circuits and Systems for Video Technology, September 2007.
ii. Recommendation ITU-T H.264 | ISO/IEC 14496-10:2009, 'Advanced Video Coding for generic audio-visual services', March 2009.
iii. Joint Scalable Video Model software, http://ip.hhi.de/imagecom_G1/savce/downloads/SVC-Reference-Software.htm
iv. ISO/IEC JTC 1/SC 29/WG 11 N9577, 'SVC Verification Test Report', Joint Video Team, Turkey, January 2007.
v. Vidyo press release, 'Vidyo unveils VidyoOne 'Telework' HD video conferencing system', New Jersey, October 2009.
vi. Radvision press release, 'Radvision to bring scalable video coding technology to SCOPIA conferencing platform', Berlin / Tel Aviv, April 2009.
vii. Joint Video Team Document JVT-AD007, 'Editors' Draft Revision to ITU-T Rec. H.264 | ISO/IEC 14496-10 Advanced Video Coding', Geneva, February 2009.
viii. Y. Chen, K. Wang, K. Ugur, M. Hannuksela, J. Lainema and M. Gabbouj, 'The Emerging MVC Standard for 3D Video Services', *EURASIP Journal on Advances in Signal Processing*, vol. 2009, Article 786015, 2009.
ix. Joint Video Team Document JVT-AC207, 'WD 3 Reference Software for MVC', Busan, October 2008.
x. Fujitsu Press Release, 'Fujitsu Launches SD Multi-Decoder LSI Supporting MPEG-2 H.264', (Tokyo), November 2008.
xi. P.A. Chou, A. Eleftheriadis, C. Herpel, C. Reader, and J. Signes, 'The MPEG-4 Systems and Description Languages: A Way Ahead in Audio Visual Information Representation', *Signal Processing: Image Communication*, vol. 9, no. 4, pp. 385–431, May 1997.
xii. E. S. Jang, J. Ohm and M. Mattavelli, 'Whitepaper on Reconfigurable Video Coding (RVC)', ISO/IEC JTC1/SC29/WG11 document N9586, Antalya, January 2008.
xiii. ISO/IEC 23002-4 'Information technology – MPEG video technologies – Part 4: Video tool library', Final Draft International Standard, August 2009.
xiv. ISO/IEC 23001-4 'Information technology – MPEG systems technologies – Part 4: Codec configuration representation', Final Draft International Standard, August 2009.
xv. I. Richardson, M. Bystrom, S. Kannangara and M. de Frutos Lopez, 'Dynamic Configuration: Beyond Video Coding Standards', IEEE International System on Chip Conference, September 2008.
xvi. I. Richardson, C. S. Kannangara, M. Bystrom, J. Philp and M. De. Frutos-Lopez, 'A Framework for Fully Configurable Video Coding,' Proc. International Picture Coding Symposium 2009, Chicago, May 2009.
xvii. I.E. Richardson, C.S. Kannangara, M. Bystrom, J. Philp and Y. Zhao, 'Fully Configurable Video Coding – A Proposed Platform for Reconfigurable Video Coding', Document M16752, ISO/IEC JTC1/SC29/WG11 (MPEG), London, July 2009.
xviii. 'Call for Evidence on High Performance Video Coding', Document N10553, ISO/IEC JTC1/SC29/WG11 (MPEG), Maui, April 2009.
xix. 'Results of Call for Evidence on High Performance Video Coding', Document W10721, ISO/IEC JTC1/SC29/WG11 (MPEG), London, July 2009.

Index

Note: Page numbers followed by *f*, *n* and *t* indicate figures, notes, and tables, respectively.

1080i video format, 19*t*, 19*f*
1080p video format, 19, 19*t*, 227, 228, 229*t*, 301, 302*f*
3D video, 302
3G, 2, 3
4:2:0 sampling, 15*f*, 16, 17*f*, 31*f*, 122, 125*f*, 126*f*, 156, 182, 184*f*, 203, 225*f*, 226*f*, 227*f*, 257*f*
4:2:2 sampling, 15, 15*f*, 16, 17, 124, 182, 184*f*, 204, 225, 226*f*, 227*f*
4:4:4 sampling, 14*f*, 15*f*, 16, 124, 185, 205, 206, 225, 226*f*, 227*f*
4CIF video format, 16, 17*t*, 301*f*
720p video format, 19, 19*f*, 19*t*, 93, 228, 229*t*, 301, 302*f*

access unit, 99, 100, 101, 102*t*, 114*t*, 116, 230, 230*f*, 231, 231*t*
Arbitrary Slice Order, 237, 238, 238*f*
arithmetic coding, 65, 67*f*, 68, 89, 179, 208, 217, 220, 253

B-skip, 165, 279
Base Layer, 289, 290, 291, 291*f*, 292, 293, 294, 297, 298, 299, 300
Baseline Profile, 93, 99*t*, 115, 115*t*, 141, 141*f*, 169, 210, 224, 261, 263, 263*f*, 267, 273, 278
basis patterns, 44, 45*f*, 87, 88*f*, 90, 90*f*
binarize, 217, 218
biprediction, 116*t*, 117*t*, 124, 162, 163*f*
Blu-Ray, 4, 98

CABAC (Context Adaptive Binary Arithmetic Coding), 92, 103*t*, 118, 124, 127*f*, 181, 208, 217, 218, 218*f*, 220, 224, 225*f*, 226*f*, 227*f*, 260*t*, 261, 265, 267*t*, 268*t*, 269*f*, 271, 273
CAVLC (Context Adaptive Variable Length Coding), 92, 103*t*, 116*t*, 118*t*, 124, 126*f*, 179, 206, 208, 210, 210*t*, 211, 211*f*, 220, 224, 225*f*, 226*f*, 227*f*, 260, 265, 266*t*, 267*t*, 268 *t*, 273
CCD (Charge Coupled Device), 8, 9
CIF video format, 16, 17*t*, 70, 93, 141, 141*f*, 165, 228, 229*t*, 261, 264, 270, 270*f*, 271, 271*f*, 291, 300, 301*f*
Coarse Grain Quality Scalability, 298
Coded Block Pattern, 96, 100*f*, 101, 103*t*, 120, 119*f*, 120*t*, 121*t*, 124, 127, 210, 212*n*, 264
Coded Picture Buffer, 84, 84*f*, 230, 230*f*, 231, 232*f*, 233, 234*f*, 236, 275*f*
colour plane coding, 206, 226*f*, 227*f*
configuration file, 257, 258*f*, 259, 260*t*, 261
conformance with H.264 standard, 6, 223–253, 235, 236, 236*f*, 237, 237*f*, 261

The H.264 Advanced Video Compression Standard, Second Edition Iain E. Richardson
© 2010 John Wiley & Sons, Ltd

Constrained Baseline Profile, 93, 169, 224
context models, 219, 219*t*, 220, 220*t*, 260

data partitioned slices, 114, 237, 243–244
DC transform, 181, 182, 203–204
Decoded Picture Buffer, 86, 101, 103*f*, 104*f*, 106, 108*f*, 111, 149, 150, 151, 162, 162*f*, 226, 230, 230*f*, 275*f*, 294
Decoder Description Syntax, 308
Differential Pulse Code Modulation, 41
Direct Mode, 122, 149, 151, 161–162, 260*t*
Discrete Cosine Transform, 30, 43, 87, 180, 181, 185
Discrete Wavelet Transform, 48s
Double Stimulus Continuous Quality Scale, 20
DVD (Digital Versatile Disk), 1, 2, 3, 3*f*, 4, 5, 16, 19, 81, 83, 97, 98, 224, 274*t*

Enhancement Layer, 289, 290, 291, 292, 294, 297, 298, 299, 300
entropy encoder, 27, 56, 57–68
Exp-Golomb codes, 208

field scan, 113, 114, 207*f*, 211
Flexible Macroblock Ordering, 238
frame / field prediction, 164
free-viewpoint video, 303
frequency dependent quantization, 204–205, 204*f*
Full Reference, 23
Fully Configurable Video Coding, 308–310, 309*f*

Group of Pictures, 169–170, 169*f*, 275, 293

H.263 standard, 26, 31, 68
H.264/AVC standard, 81, 82, 91, 92*t*, 99, 99*t*, 180, 207, 224, 252, 290, 293
Hadamard Transform, 181, 182, 203, 220, 284
hierarchical prediction, 170, 273, 293*f*
High Definition, 1, 2, 5, 18–19, 19*f*, 93, 98, 223, 224, 225, 227, 228, 287, 290, 301
High Profile, 119f, 120, 124, 139*t*, 148, 181, 182, 204, 217, 224, 225, 226f, 237, 248, 287, 293, 310
Huffman coding, 58–62, 62*t*, 63, 65, 68
Human Visual System, 13, 20, 204
Hypothetical Reference Decoder, 92*t*, 223, 230, 230*f*, 249*t*, 250*t*, 253

Instantaneous Decoder Refresh, 101, 114
integer transform, 87, 120, 179, 179, 181, 185, 198–199, 199*t*, 200*t*, 211, 224
inter prediction, 6, 27, 40, 85, 85*n*, 87*f*, 91, 101, 120, 122, 132, 137, 149–170, 167*f*, 168*f*, 182, 182, 225, 264, 265, 271, 276, 297
interlaced video, 11, 11*f*, 16, 19*t*, 111, 207, 224, 260*t*
International Standards Organisation, 4
International Telecommunications Union, 4
Intra PCM, 121–122
intra prediction, 26, 27, 32, 38, 39*f*, 85, 86*f*, 91, 95, 103*t*, 116*t*, 117*t*, 120*t*, 137, 138–149, 171*n*, 181, 185, 203, 206, 224, 243, 264, 276, 297, 310
ISO Media File Format, 247
ITU-R 601, 7

Joint Model reference software, 255, 256, 285

Lagrange, 281
licensing of video coding standards, 4, 223, 248–253
loop filter, 171–176, 260*t*, 265, 266*t*, 267*t*, 268*t*
lossless predictive coding, 205–206, 226*f*, 227*f*

macroblock, 31–32, 31*f*, 57–58, 69, 70, 74*f*, 75*f*, 77, 84, 85, 86*f*, 87*f*, 91, 94, 94*f*, 95, 96*f*, 101, 103*t*, 111, 112, 113*f*, 117, 117*t*, 118, 119–135, 137–138, 142*f*, 146*f*, 149, 158, 159, 160–167, 171*n*,

172f, 173*f*, 181, 184*f*, 210*t*, 227, 228, 238, 239*t*, 239*f*, 264, 274*f*, 276, 279, 280, 282, 284
Macroblock Adaptive Frame Field Coding, 112, 113*f*, 164, 260*t*
macroblock partition, 117t, 130, 132, 149, 157, 158f, 158t, 210t, 264, 279, 282, 297
Main Profile, 93, 115*t*, 224, 225*f*, 226, 261, 264, 264*f*, 268, 273
Medium Grain Quality Scalability, 298, 298*f*
mode selection, 166, 255, 256, 260*t*, 261, 279, 281–283, 284, 285
motion compensated prediction, 27, 31–32, 35, 69, 70, 103*t*, 111, 162–164, 162*f*, 165, 224, 241, 243, 247, 249*t*, 252, 264, 281
motion compensation, 26, 30, 31, 32, 33*f*, 35–38, 41, 72*f*, 85*n*, 155, 159, 165, 175, 266*t*, 267*t*, 268*t*
motion estimation, 26, 30–32, 32*f*, 35, 36*f*, 69, 72, 85*n*, 260*t*, 261
motion vector prediction, 60*f*, 158–161, 297
Moving Picture Experts Group (MPEG), 5, 310
MPEG-2 standard, 26, 31, 68, 83, 99, 97*f*, 169, 180, 246, 246*f*, 250, 251, 252, 302, 306
MPEG-LA, 252, 253
multiple reference frames, 130, 131*t*, 170, 267
Multiview Video Coding, 287, 302–305

Network Abstraction Layer, 100, 100*f*, 102*t*, 114, 244, 245*f*
No Reference, 23
NTSC, 17, 250*t*

PAL, 12, 17, 250*t*
Peak Signal to Noise Ratio, 21
Picture Adaptive Frame Field Coding, 142*f*
Picture Order Count, 104, 118, 118*t*
Picture Parameter Set, 100, 102*t*, 114*t*, 115, 116*t*, 116*f*, 205, 244
progressive video, 12, 111

QCIF video format, 16, 17*t*, 37*t*, 53, 115, 115*t*, 141, 142*f*, 227, 228, 229*t*, 239*f*, 259, 265, 266*f*, 267*f*, 268*f*, 269*f*, 270*f*, 271, 273, 273*f*, 278, 279*f*
Quality Scalability, 290, 291, 291*f*, 294, 295*f*, 298–299, 299*f*, 300, 301*f*
quantization, 26, 50–52, 53*f*, 54, 58, 78, 87, 88f, 89*n*, 90, 96, 102*t*, 174, 179–206, 260*t*
quantizer step size, 58, 76, 191, 192*f*, 198, 202, 204, 277

rate control, 247, 255, 260*t*, 265*t*, 266*t*, 267*t*, 268, 268*t*, 269*f*, 274–278
Rate Distortion Optimization, 265, 266, 267, 268*t*
Raw Byte Sequence Payload, 101, 102*t*, 244, 245*f*, 248
Real-Time Protocol, 246
Reconfigurable Video Coding, 287, 307–308, 307*f*
reconstruction, 48, 69, 70, 83, 91, 91*f*
Reduced Reference, 23
redundant slices, 224, 225*f*, 237, 260*t*
rescaling matrix, 189, 200–202
residual, 27, 28, 30, 32, 33*f*, 34*f*, 35–38, 40–43, 49, 53, 54, 54*f*, 69, 70, 72*f*, 74*f*, 75*f*, 78*f*, 84, 84*f*, 85, 86*f*, 87, 87*f*, 90, 96*f*, 97*f*, 101, 103*t*, 121, 124, 125*f*, 126*f*, 127*f*, 141, 143*f*, 165, 180, 193, 210, 243, 244, 250, 281, 297
RGB colour space, 12–13, 14
run-level encoding, 56

Scalable Video Coding, 92*t*, 247, 287, 288–302
scalar quantization, 50
Sequence Parameter Set, 93, 100, 101, 102*t*, 105, 111, 112, 114*t*, 115, 115*t*, 206, 226, 244, 248, 262*t*
SI-slice, 243
simulcast, 288–289, 299, 300, 300*f*, 301
SP-slice, 241, 242*t*, 242*f*, 243, 243*f*, 244*f*
spatial sampling, 9

Spatial Scalability, 290, 291*f*, 294, 296*f*, 298, 301, 302*f*
SSIM, 23
stereoscopic video, 287, 302
stream analyzer, 263, 263*f*, 264*f*
sub-macroblock partition, 122, 123, 149, 150, 157, 158*f*, 158*t*, 159, 162, 166
sub-pixel motion compensation, 35–38
sub-pixel prediction, 152*f*
Sum of Absolute Errors, 36, 145
Sum of Absolute Transformed Differences, 282, 284
Sum of Squared Distortion, 282
Supplemental Enhancement Information, 92*t*, 102*t*, 114*t*, 248

temporal sampling, 8*f*, 9–11, 26
Temporal Scalability, 170, 290, 291*f*, 292, 292*t*, 293, 301, 301*f*
trailing bits, 101, 102*t*, 245*f*

Universal Video Decoder, 308, 310

VC-1 standard, 26, 68, 180, 306, 310
vector quantization, 52, 53*f*
video coding, 4, 7, 25–79, 81, 82, 82*f*, 83*f*, 89, 92*t*, 223, 247, 248, 252, 253, 262, 268, 287, 288–310
Video Coding Experts Group, 5, 310
Video Coding Layer, 100, 114
video coding patents, 250–251, 252
video compression, 2–6, 16, 21, 24, 25, 43, 51, 81, 83, 85, 223, 237, 248, 250, 251, 251*f*, 252, 287, 288, 310
Video Quality Experts Group, 23
Video Usability Information, 94*t*, 225, 248, 250*t*
videoconferencing, 16, 81, 82, 169, 247, 274*t*, 302
weighted prediction, 116*t*, 137, 149, 163–164, 260*t*

x264 software, 261, 261*n*, 262, 262*f*, 263

YCrCb colour space, 13–16
Youtube, 5

zig zag scan, 54, 112