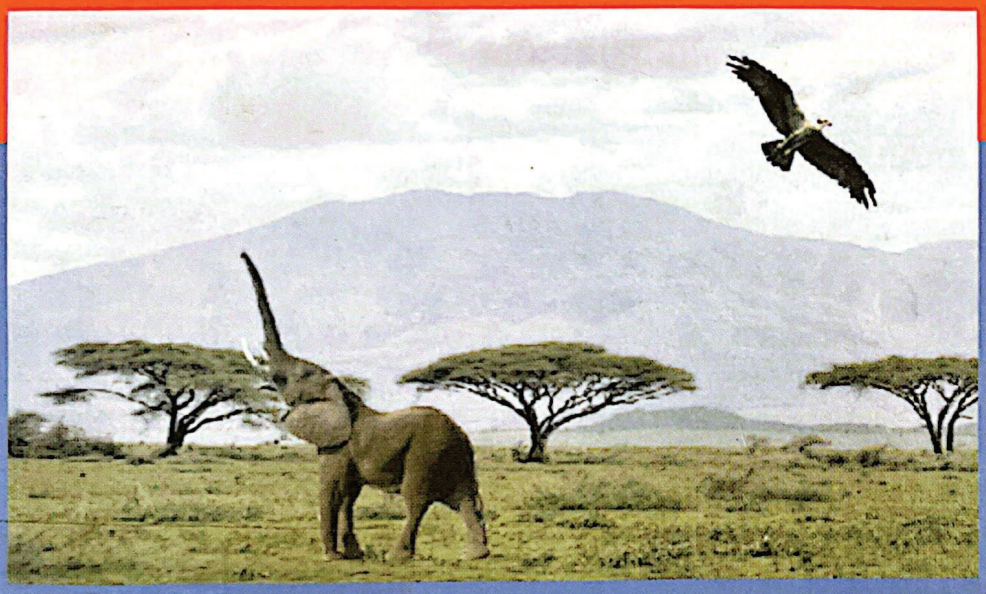
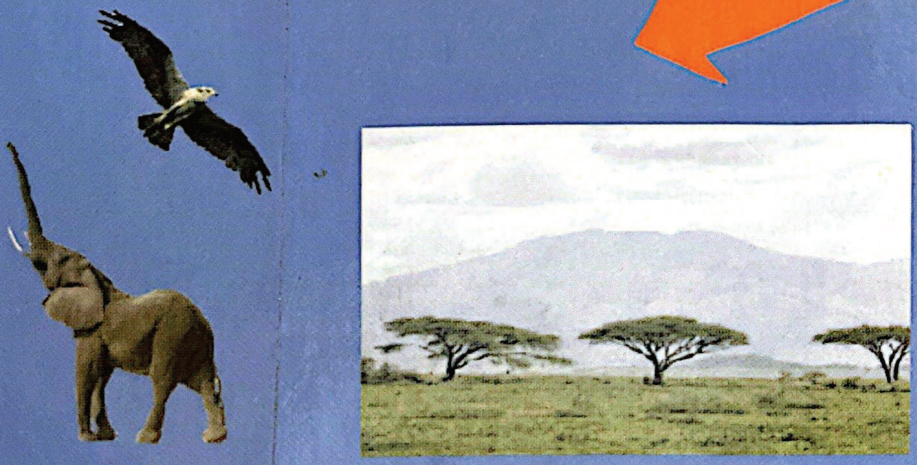
IEE TELECOMMUNICATIONS SERIES 42

# Video coding

## an introduction to standard codecs

Mohammed Ghanbari





The Institution of Electrical Engineers

ELEMENT-1018
IPR2025-01108, IPR2025-01109

Published by: The Institution of Electrical Engineers, London,
United Kingdom

© 1999: The Institution of Electrical Engineers

This publication is copyright under the Berne Convention and the Universal Copyright Convention. All rights reserved. Apart from any fair dealing for the purposes of research or private study, or criticism or review, as permitted under the Copyright, Designs and Patents Act, 1988, this publication may be reproduced, stored or transmitted, in any forms or by any means, only with the prior permission in writing of the publishers, or in the case of reprographic reproduction in accordance with the terms of licences issued by the Copyright Licensing Agency. Inquiries concerning reproduction outside those terms should be sent to the publishers at the undermentioned address:

The Institution of Electrical Engineers,
Michael Faraday House,
Six Hills Way, Stevenage,
Herts. SG1 2AY, United Kingdom

While the author and the publishers believe that the information and guidance given in this work are correct, all parties must rely upon their own skill and judgment when making use of them. Neither the author nor the publishers assume any liability to anyone for any loss or damage caused by any error or omission in the work, whether such error or omission is the result of negligence or any other cause. Any and all such liability is disclaimed.

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

**British Library Cataloguing in Publication Data**

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

**ISBN 0 85296 762 4**

Printed in England by Short Run Press Ltd., Exeter

# Contents


**Preface** xi

**Chapter 1 History of video coding** 1

**Chapter 2 Video basics** 7
2.1 Analogue video 7
2.1.1 Scanning 7
2.1.2 Colour components 8
2.2 Digital video 9
2.3 Image format 11
2.3.1 SIF images 11
2.3.2 Conversion from SIF to CCIR-601 format 14
2.3.3 CIF image format 15
2.3.4 Sub-QCIF, QSIF, QCIF 16
2.3.5 HDTV 17
2.3.6 Conversion from film 18
2.3.7 Temporal resampling 18
2.4 Picture quality assessment 18
References 19

**Chapter 3 Principles of video compression** 20
3.1 Spatial redundancy reduction 21
3.1.1 Predictive coding 21
3.1.2 Transform coding 22
3.1.3 Fast DCT transform 24
3.2 Quantisation of DCT coefficients 25
3.3 Temporal redundancy reduction 27
3.3.1 Motion estimation 28
3.3.2 Fast motion estimation 29
3.3.3 Hierarchical motion estimation 31
3.4 Variable length coding 32
3.4.1 Huffman coding 33
3.4.2 Arithmetic coding 34

3.4.2.1 Principles of arithmetic coding 35
3.4.2.2 Binary arithmetic coding 38
3.4.2.3 An example of binary arithmetic coding 43
3.4.2.4 Adaptive arithmetic coding 44
3.5 A generic interframe video codec 46
3.5.1 Interframe loop 46
3.5.2 Motion estimator 46
3.5.3 Inter/intra switch 47
3.5.4 DCT 47
3.5.5 Quantiser 47
3.5.6 Variable length coding 47
3.5.7 IQ and IDCT 47
3.5.8 Buffer 48
References 48

**Chapter 4 Coding of still pictures (JPEG)** **50**
4.1 Lossless compression 51
4.2 Lossy compression 52
4.2.1 Baseline sequential mode compression 52
4.2.2 Run-length coding 55
4.2.2.1 Coding of DC coefficients 55
4.2.2.2 Coding of AC coefficients 56
4.2.2.3 Entropy coding 58
4.2.3 Extended DCT-based process 58
4.2.4 Hierarchical mode 60
4.2.5 Extra features 63
4.3 JPEG-2000 63
4.3.1 Criteria for next generation compression 64
4.3.2 Methods of compression 67
References 67

**Chapter 5 Coding for video conferencing (H.261)** **69**
5.1 Video format and structure 70
5.2 Video source coding algorithm 72
5.2.1 Prediction 73
5.2.2 MC/NO_MC decision 73
5.2.3 Inter/Intra decision 74
5.2.4 Forced updating 75
5.3 Other types of macroblocks 75
5.3.1 Addressing of macroblocks 76
5.4 Quantisation and coding 78
5.4.1 Two-dimensional variable length coding 78
5.5 Loop filter 82
5.6 Rate control 84
References 86

**Chapter 6 Coding of moving pictures for digital storage media (MPEG-1)** **87**
6.1 Systems coding outline 88
6.1.1 Multiplexing elementary streams 89
6.1.2 Synchronisation 89
6.2 Preprocessing 90
6.2.1 Picture re-ordering 91
6.3 Video structure 92
6.3.1 Group of pictures (GOP) 92
6.3.2 Picture 93
6.3.3 Slice 93
6.3.4 Macroblock 95
6.3.5 Block 95
6.4 Encoder 96
6.5 Quantisation weighting matrix 98
6.6 Motion estimation 99
6.6.1 Larger search range 100
6.6.2 Motion estimation with half pixel precision 101
6.6.3 Bi-directional motion estimation 103
6.7 Coding of pictures 104
6.7.1 I-picture 104
6.7.2 P-pictures 105
6.7.3 B-pictures 106
6.7.4 D-pictures 107
6.8 Video buffer verifier 107
6.8.1 Buffer size and delay 109
6.8.2 Rate control and adaptive quantisation 110
6.9 Decoder 111
6.9.1 Decoding for fast play 112
6.9.2 Decoding for pause and step mode 113
6.9.3 Decoding for reverse play 113
6.10 Postprocessing 114
6.10.1 Editing 114
6.10.2 Resampling and up-conversion 116
References 116

**Chapter 7 Coding of high quality moving pictures (MPEG-2)** **117**
7.1 MPEG-2 systems 118
7.2 Level and profile 121
7.3 How does the MPEG-2 video encoder differ from MPEG-1? 122
7.3.1 Major differences 122
7.3.2 Minor differences 123
7.3.3 MPEG-1 and MPEG-2 syntax differences 124
7.4 MPEG-2 non-scalable coding modes 124

7.4.1 Frame prediction for frame pictures 125
7.4.2 Field prediction for field pictures 125
7.4.3 Field prediction for frame pictures 127
7.4.4 Dual-prime for P-pictures 127
7.4.5 16×8 motion compensation for field pictures 129
7.4.6 Restrictions on field pictures 129
7.4.7 Motion vectors for chrominance components 130
7.4.8 Concealment motion vectors 130
7.5 Scalability 131
7.5.1 Data partitioning 133
7.5.2 SNR scalability 136
7.5.3 Spatial scalability 141
7.5.4 Temporal scalability 144
7.5.5 Hybrid scalability 146
7.5.5.1 Spatial and temporal hybrid scalability 146
7.5.5.2 SNR and spatial hybrid scalability 147
7.5.5.3 SNR and temporal hybrid scalability 148
7.5.5.4 SNR, spatial and temporal hybrid scalability 148
7.5.6 Applications of scalability 149
References 151

**Chapter 8 Video coding for low bit rate communications (H.263) 152**
8.1 How does H.263 differ from H.261 and MPEG-1? 153
8.1.1 Coding of H.263 coefficients 153
8.1.2 Coding of motion vectors 153
8.1.3 Source pictures 155
8.1.4 Picture layer 156
8.2 Switched multipoint 157
8.2.1 Freeze picture request 157
8.2.2 Fast update request 158
8.2.3 Freeze picture release 158
8.2.4 Continuous presence multipoint 158
8.3 Optional modes of H.263 158
8.3.1 Unrestricted motion vector mode 158
8.3.2 Advanced prediction mode 159
8.3.2.1 Four motion vectors per macroblock 159
8.3.2.2 Overlapped motion compensation 160
8.3.3 Syntax-based arithmetic coding mode 162
8.3.4 PB-frames mode 162
8.3.4.1 Macroblock type 163
8.3.4.2 Motion vectors for B-pictures in PB-frames 163
8.3.5 Prediction for a B-block in PB-frame 164
8.4 Coding efficiency of H.263 165
8.5 Video coding for very low bit rates 167
8.5.1 Scope and goals of H.263+ 168

8.5.2 Advanced intra coding 168
8.5.3 Advanced inter coding with switching between two VLC tables 170
8.5.4 Improved PB-frames mode 171
8.6 Deblocking filter mode 172
8.7 Protection against error 174
8.7.1 Error correction 174
8.7.2 Error resilience 174
8.7.3 Decoder with error concealment 175
8.8 Video over ATM networks 177
8.8.1 Loss concealment 177
8.8.2 Effect of packetisation 179
8.9 Buffer regulation 182
8.10 Scalability 184
8.10.1 Temporal scalability 184
8.10.2 SNR scalability 185
8.10.3 Spatial scalability 186
8.10.4 Multilayer scalability 186
8.10.5 Transmission order of pictures 188
8.11 Advanced video coding (H.263L) 189
References 190

**Chapter 9 Content-based video coding (MPEG-4) 191**
9.1 Video object plane (VOP) 194
9.1.1 Coding of objects 195
9.1.2 Encoding of VOPs 195
9.1.3 Formation of VOP 196
9.2 Image segmentation 197
9.2.1 Segmentation 197
9.2.2 Object segmentation 198
9.3 Shape coding 200
9.3.1 Coding of binary alpha planes 200
9.3.2 Chain code 200
9.3.3 Quad-tree coding 202
9.3.4 Modified modified Reed (MMR) 205
9.3.5 Context based arithmetic coding 207
9.3.5.1 Size conversion 208
9.3.5.2 Generation of context index 209
9.3.6 Grey scale shape coding 212
9.4 Motion estimation and compensation 213
9.5 Texture coding 214
9.5.1 Shape adaptive DCT 214
9.6 Coding of synthetic objects 215
9.7 Coding of still images 218
9.7.1 Discrete wavelet transform 218
9.7.2 Higher order systems 224

9.7.3 Coding of the lowest band 226
9.7.4 Zero-tree coding of higher bands 226
9.7.4.1 Quantisation by successive approximation 227
9.7.4.2 Similarities among the bands 228
9.7.4.3 Embedded zero-tree wavelet (EZW) algorithm 228
9.7.5 Shape adaptive wavelet transform 231
9.8 Video coding with the wavelet transform 232
9.8.1 Virtual zero-tree (VZT) algorithm 233
9.8.2 Set partioning in hierarchical trees (SPIHT) 235
9.9 MPEG-7: Multimedia content description interface 239
References 242

**Appendix A: A 'C'-program for the fast discrete cosine transform 245**

**Appendix B: Huffman tables for the DC and AC coefficients of the JPEG base-line encoder 248**

**Appendix C: Huffman tables for quad-tree shape coding 251**

**Appendix D: Frequency tables for the CAE encoding of binary shapes 253**

**Index 257**

*Chapter 3*

# Principles of video compression

The statistical analysis of video signals indicates that there is a strong correlation both between successive picture frames and within the picture elements themselves. Theoretically decorrelation of these signals can lead to bandwidth compression without significantly affecting image resolution. Moreover, the insensitivity of the human visual system to loss of certain spatio-temporal visual information can be exploited for further reduction. Hence, subjectively lossy compression techniques can be used to reduce video bit rates while maintaining an acceptable image quality.

For coding still images, only the spatial correlation is exploited. Such a coding technique is called *Intraframe* coding and is the basis for JPEG coding. If temporal correlation is exploited as well, then it is called *Interframe* coding. Interframe predictive coding is the main coding principle that is used in all standard video codecs, such as H.261, H.263, MPEG-1, 2 and 4. It is based on three fundamental redundancy reduction principles:

1. Spatial redundancy reduction: to reduce spatial redundancy among the pixels within a picture (similarity of pixels, within the frames), by employing some data compressors, such as transform coding.
2. Temporal redundancy reduction: to remove similarities between the successive pictures, by coding their differences.
3. Entropy coding: to reduce the redundancy between the compressed data symbols, using variable length coding techniques.

A detailed description of these redundancy reduction techniques is given in the following sections.

## 3.1 Spatial redundancy reduction

### *3.1.1 Predictive coding*

In the early days of image compression, both signal processing tools and storage devices were scarce resources. At the time, a simple method for redundancy reduction was to predict the value of pixels based on the values previously coded, and code the prediction error. This method is called *Differential Pulse Code Modulation* (DPCM). Figure 3.1 shows a block diagram of a DPCM codec, where the differences between the incoming pixels from the predictions in the predictor are quantised and coded for transmission. At the decoder the received error-signal is added to the prediction to reconstruct the signal. If the quantiser is not used it is called *loss-less* coding, and the compression relies on the entropy coder, which will be explained later.

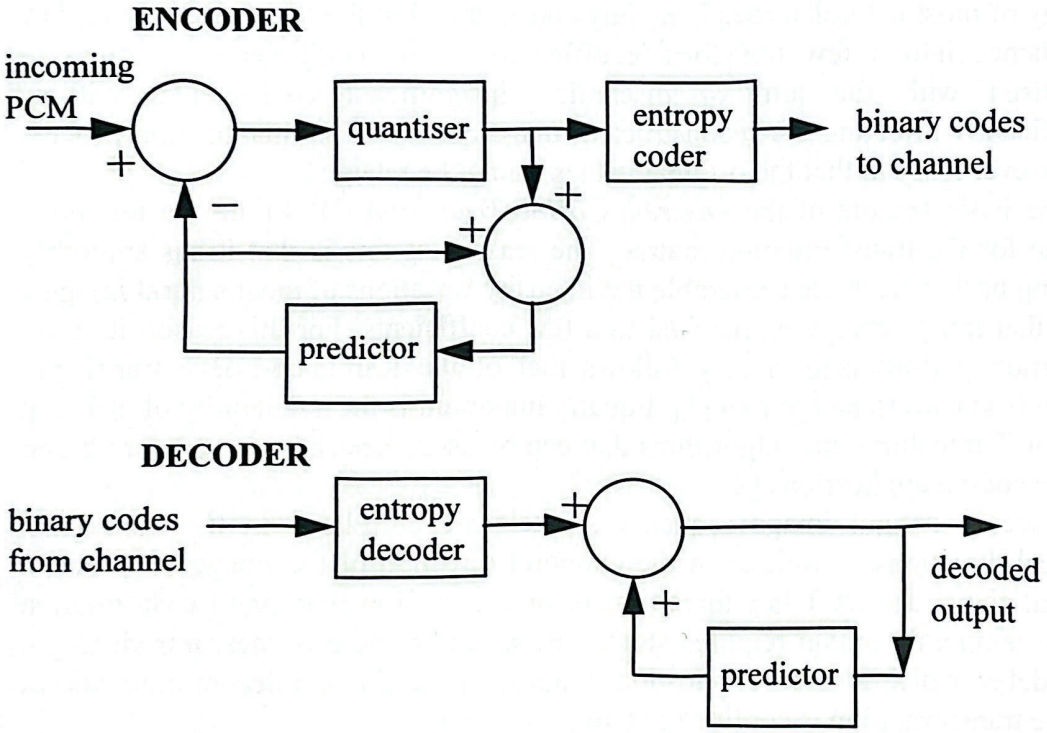



*Figure 3.1 Block diagram of a DPCM codec*

Best predictions are those from the neighbouring pixels, either from the same frame or pixels from the previous frame, or their combinations. The former is called *Intraframe predictive coding* and the latter is *Interframe predictive coding*. Their combination is called *Hybrid predictive coding*.

It should be noted that, no matter what prediction is used, every pixel is predictively coded. The minimum number of bits that can be assigned to each prediction error is 1 bit. Hence this type of coding is not suitable for low bit rate

video coding. Lower bit rates can be achieved if a group of pixels are coded together, such that the average bit per pixel can be less than 1 bit. Block transform coding is most suitable for this purpose. Despite this, DPCM is still used in video compression. For example interframe DPCM has lower coding latency than interframe block coding. Also, DPCM might be used in coding of motion vectors, or block addresses. If motion vectors in a moving object move in the same direction, coding of their differences will reduce the motion vector information. Of course, the coding would be loss-less.

### *3.1.2 Transform coding*

Transform domain coding is mainly used to remove the spatial redundancies in images by mapping the pixels into a transform domain prior to data reduction. The strength of transform coding in achieving data compression is that the image energy of most natural scenes is mainly concentrated in the low frequency region, and hence into a few transform coefficients. These coefficients can then be quantised with the aim of discarding insignificant coefficients, without significantly affecting the reconstructed image quality. This quantisation process is however lossy in that the original values cannot be retained.

The basis vectors of the *Discrete Cosine Transform* (DCT) are the universal choice for the transformation matrix. The reason for this is that it has smoothly varying basis vectors that resemble the intensity variations of most natural images, such that image energy is matched to a few coefficients. For this reason its rate-distortion performance closely follows that of the Karhunen-Loève transform, which is known to be optimal [1]. Equally important is the availability of efficient fast DCT transformation algorithms that can be used, especially in software based image coding applications [2].

Since in natural image sequences, pixels are correlated in the horizontal, vertical directions as well as in the temporal direction of the image sequence, a natural choice for DCT is a three-dimensional one. However, any transformation in the temporal domain requires storage of several picture frames, introducing a long delay, which restricts application of transform coding in telecommunications. Hence transformation is confined to two dimensions.

A two-dimensional DCT is a separable process that is implemented using two one-dimensional DCTs: one in the horizontal direction followed by one in the vertical. For a block of $M \times N$ pixels, the forward one-dimensional transform of $N$ pixels is given by

$$F(u) = \frac{1}{\sqrt{N}} C(u) \sum_{x=0}^{N-1} f(x) \cos\left(\frac{\pi(2x+1)u}{2N}\right) \quad u = 0,1,\ldots,N-1 \tag{3.1}$$

where $C(u) = \frac{1}{\sqrt{2}}$ for u = 0

$C(u) = 1$ otherwise

$f(x)$ represents the intensity of the $x$th pixel, and $F(u)$ represents the $N$ one-dimensional transform coefficients. The inverse one-dimensional transform is thus defined as

$$f(x) = \frac{1}{\sqrt{N}} \sum_{u=0}^{N-1} C(u)F(u)\cos\left(\frac{\pi(2x+1)u}{2N}\right) \quad x = 0,1,\ldots,N-1 \tag{3.2}$$

Note that the $1/\sqrt{N}$ normalisation factor is used to make transformation orthonormal. That is, the energy in both pixel and transform domains is to be equal. In the standard codecs the normalisation factor in the DCT domain is defined as 1/2. This gives the DCT coefficients in the range of –2047 to +2047. The normalisation factor in the pixel domain is then adjusted accordingly (e.g. it becomes $2/N$).

To derive the final two-dimensional transform coefficients, $N$ sets of one-dimensional transforms of length $M$ are taken over the one-dimensional transform coefficients of similar frequency in the vertical direction:

$$F(u,v) = \frac{1}{\sqrt{M}} C(v) \sum_{y=0}^{M-1} F(u,y)\cos\left(\frac{\pi(2y+1)}{2M}\right) \quad v = 0,1,\ldots,M-1 \tag{3.3}$$

where $C(v)$ is defined similarily to $C(u)$.

Thus a block of $MN$ pixels are transformed into $MN$ coefficients. The $F(0,0)$ coefficient represents the DC value of the block. Coefficient $F(0,1)$, which is the DC value of all the first one-dimensional AC coefficients, represents the first AC coefficient in the horizontal direction of the block. Similarly, $F(1,0)$, which is the first AC coefficient of all one-dimensional DC values, represents the first AC coefficient in the vertical direction, and so on.

In practice $M$=$N$=8, such that a two-dimensional transform of $8 \times 8 = 64$ pixels results in 64 transform coefficients. The choice of such a block size is a compromise between the compression efficiency and the blocking artefacts of coarsely quantised coefficients. While larger block sizes have good compression efficiency, the blocking artefacts are subjectively very annoying. At the early stage of standardisation of video codecs, the block sizes were made optional at $4 \times 4$, $8 \times 8$ and $16 \times 16$. Now the block size in standard codecs is $8 \times 8$.

### *3.1.3 Fast DCT transform*

To calculate transform coefficients, every one-dimensional forward or inverse transformation requires eight multiplications and seven additions. This process is repeated for 64 coefficients, both in the horizontal and vertical directions. Since software based video compression is highly desirable, methods of reducing such a huge computational burden are highly desirable.

The fact that DCT is a type of discrete Fourier transform, with the advantage of all-real coefficients, then one can use a fast transform, similar to the fast Fourier transform, to calculate transform coefficients with complexity proportional to $N \log_2 N$, rather than $N^2$. Figure 3.2 shows a butterfly representation of the fast DCT [2]. Intermediate nodes share some of the computational burden, hence reducing the overall complexity. In the figure, p[0]-p[7] are the inputs to the forward DCT and b[0]-b[7] are the transform coefficients. The inputs can be either the eight pixels for the source image, or eight transform coefficients of the first stage of the one-dimensional transform. Similarly, for inverse transformation, b[0]-b[7] are the inputs to the IDCT, and p[0]-p[7] are the outputs. A 'C' language programme for fast forward DCT is given in Appendix A. In this program, some modifications to the butterfly matrices are made to trade off the number of additions for multiplications, since multiplications are more computationally intensive than additions. A similar program can be written for the inverse transform.

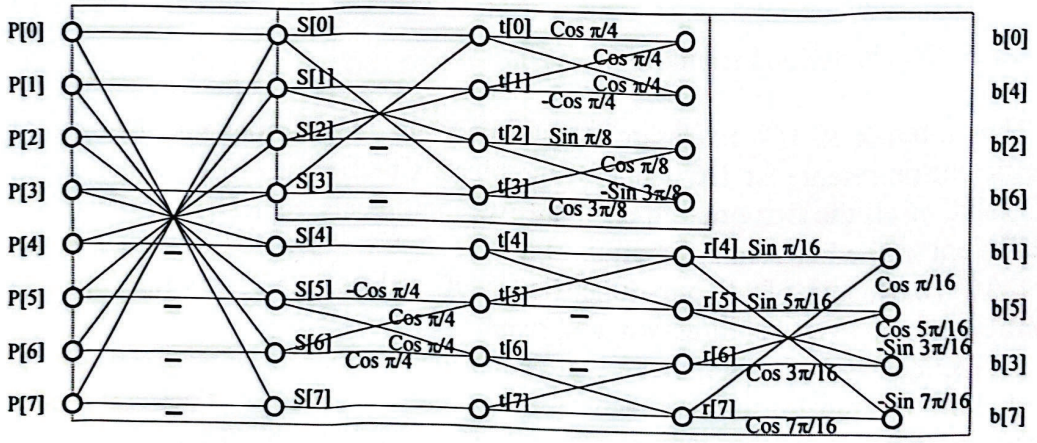



*Figure 3.2 A fast DCT flow chart*

## 3.2 Quantisation of DCT coefficients

The domain transformation of the pixels does not actually yield any compression. A block of 64 pixels is transformed into 64 coefficients. Due to the orthonormality of transformation, the energy in both the pixel and the transform domains are equal, hence no compression is achieved. However transformation causes the significant part of the image energy to be concentrated at the lower frequency components, with the majority of the coefficients having little energy. It is the quantisation and variable length coding of the DCT coefficients that lead to bit rate reduction. Moreover by exploiting the human eye's characteristics, which are less sensitive to picture distortions at higher frequencies, one can apply even coarser quantisation at these frequencies, to give greater compression. Coarser quantisation step sizes force more coefficients to zero and as a result, more compression is gained, but of course the picture quality deteriorates accordingly.

The class of quantiser that has been used in all standard video codecs is based around the so called *uniform threshold quantiser* (UTQ). It has equal step sizes with reconstruction values pegged to the centriod of the steps. This is illustrated in Figure 3.3.

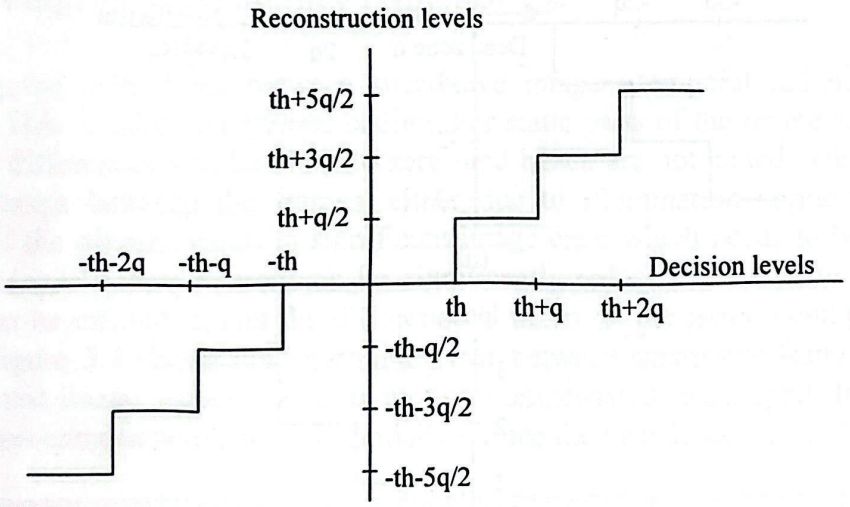



*Figure 3.3 Quantisation characteristics*

The two key parameters that define a UTQ are the threshold value, *th*, and the step size, *q*. The centroid value is typically defined mid-way between quantisation intervals. Note that, although AC transform coefficients have non-uniform characteristics, and hence can be better quantised with non-uniform quantiser step sizes (the DC coefficient has a fairly uniform distribution), bit rate control would be easier if they were quantised linearly. Hence, a key property of UTQ is that the step sizes can be easily adapted to facilitate rate control.

A further two sub-classes of UTQ can be identified within the standard codecs, namely those with and without a *dead zone*. They are illustrated in Figure 3.4 and

will be hereafter abbreviated as UTQ-DZ and UTQ respectively. The term dead zone commonly refers to the central region of the quantiser, whereby the coefficients are quantised to zero.

Typically, UTQ is used for quantising intraframe DC, $F(0,0)$, coefficients, while UTQ-DZ is used for the AC and the DC coefficients of interframe prediction error. This is intended primarily to cause more non-significant AC coefficients to become zero, so increasing the compression. In UTQ, coefficients $F(u,v)$ are quantised by their division to the quantiser step size, $q$, with rounding towards the nearest integer.

$$I(u,v) = \frac{F(u,v) \pm q}{2q} \tag{3.4}$$

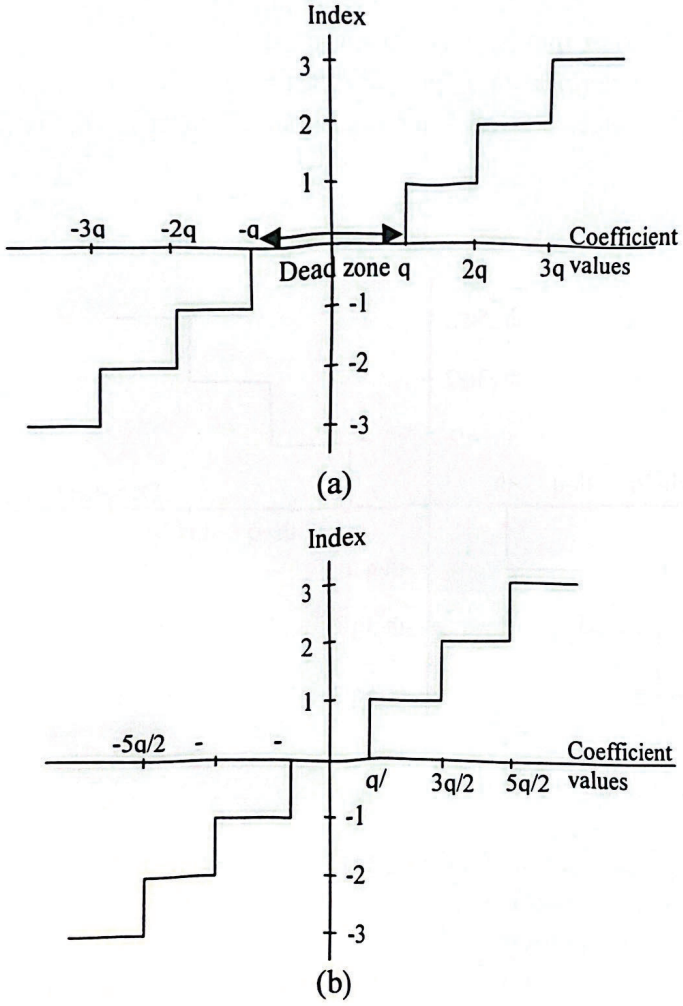



*Figure 3.4 Uniform quantisers (a) with and (b) without dead zone*

Such a ratio, $I(u,v)$, is called the *quantisation index*. It is the quantisation index that is sent to the decoder, since it has a much smaller entropy than the quantised

coefficient. At the decoder, the reconstructed coefficients, $F^q(u,v)$, after inverse quantisation are given by

$$F^q(u,v) = I(u,v) \times 2q \tag{3.5}$$

Quantisation indices employing UTQ-DZ are obtained by using integer division which will truncate the resultant towards zero:

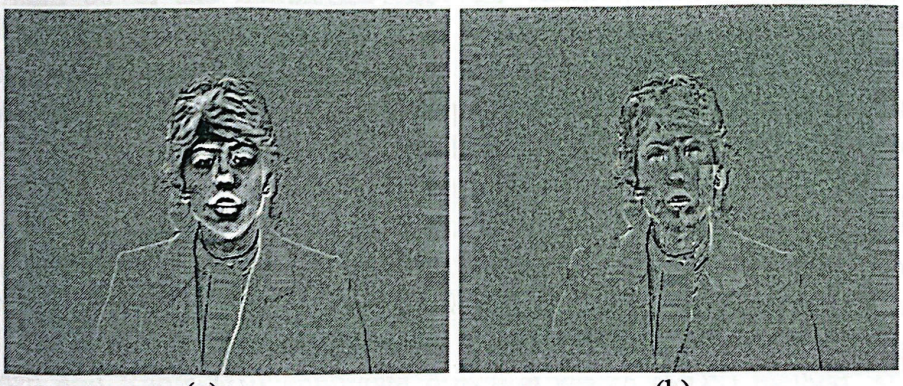
$$I(u,v) = \frac{F(u,v)}{2q} \tag{3.6}$$

In the inverse quantisation, depending on the polarity of the index, an addition or subtraction of half the quantisation step is required to deliver the centroid representation:

$$F^q(u,v) = \{2I(u,v) \pm 1\} \times q \tag{3.7}$$

## 3.3 Temporal redundancy reduction

By using the differences between successive images, temporal redundancy is reduced. This is called *interframe* coding. For static parts of the image sequence, temporal differences will be close to zero, and hence are not coded. Those parts which change between the frames, either due to illumination variation or to motion of the objects, result in significant image error which needs to be coded. Image changes due to motion can be significantly reduced if the motion of the object can be estimated, and the difference is taken on the motion compensated image. Figure 3.5 shows the interframe error between successive frames of the "Claire" test image sequence and its motion compensated counterpart. It is clear that motion compensation can substantially reduce the interframe error.



(a) (b)

*Figure 3.5 (a) Interframe and (b) motion compensated interframe pictures*

### *3.3.1 Motion estimation*

To carry out motion compensation, the motion of the moving objects has to be estimated first. This is called *motion estimation*. The commonly used motion estimation technique in all the standard video codecs is the *Block Matching Algorithm* (BMA). In a typical BMA, a frame is divided into blocks of $M \times N$ pixels or, more usually, square blocks of $N^2$ pixels [3]. Then, for a maximum motion displacement of $w$ pixels per frame, the current block of pixels is matched against a corresponding block at the same co-ordinates but in the previous frame, within the square window of width $N + 2w$ (Figure 3.6).

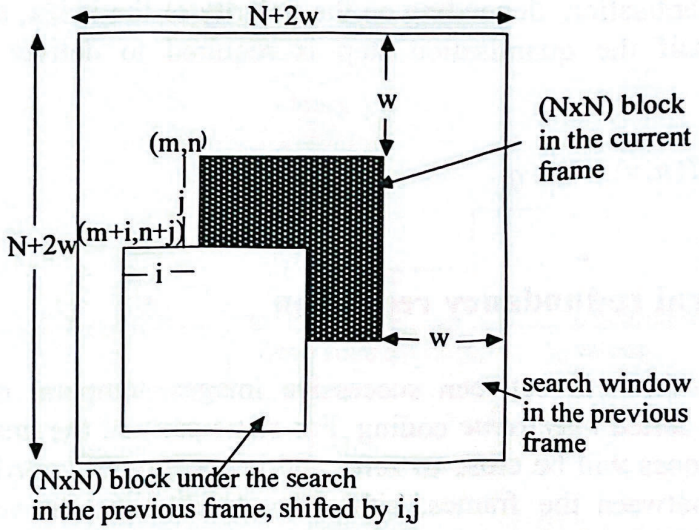



*Figure 3.6 The current and previous frames in a search window*

The best match on the basis of a matching criterion yields the displacement.

Various measures such as the crosscorrelation function (CCF), mean squared error (MSE) and mean absolute error (MAE) can be used in the matching criterion [4–6]. For the best match, in the CCF the correlation has to be maximised, whereas in the latter two the distortion must be minimised. In practical coders both MSE and MAE are used, since it is believed that CCF would not give good motion tracking, especially when the displacement is not large [6]. The matching functions of the type MSE and MAE are defined as:

for MSE:

$$M(i,j) = \frac{1}{N^2} \sum_{m=1}^{N} \sum_{n=1}^{N} \left( f(m,n) - g(m+i,n+j) \right)^2, \quad -w \le i,j \le w \tag{3.8}$$

and for MAE:

$$M(i,j) = \frac{1}{N^2} \sum_{m=1}^{N} \sum_{n=1}^{N} \left| f(m,n) - g(m+i,n+j \right|, \quad -w \le i,j \le w \tag{3.9}$$

where $f(m,n)$ represents the current block of $N^2$ pixels at co-ordinates $(m,n)$ and $g(m+i, n+j)$ represents the corresponding block in the previous frame at new co-ordinates $(m+i, n+j)$.

To locate the best match by full search, $(2w+1)^2$ evaluations of the matching criterion are required. To reduce processing cost, MAE is preferred to MSE, and hence is used in all the video codecs. However, for each block of $N^2$ pixels we still need to carry out $(2w+1)^2$ tests, each with almost $2N^2$ additions and subtractions. This is still far from being suitable for implementation of BMA in software based codecs. In fact studies by Downton [7] have shown that in the H.261 and MPEG-1 codecs, motion estimation comprises almost 60-70% of the codec complexity. Hence fast motion estimation techniques are highly desirable.

### *3.3.2 Fast motion estimation*

In the past two decades a number of fast search methods for motion estimation have been introduced to reduce the computational complexity of BMA. The basic principle of these methods is that the number of search points can be reduced, by selectively checking only a small number of specific points, assuming that the distortion measure monotonically decreases towards the best matched point. Jain and Jain [6] were the first to use a two-dimensional logarithmic (TDL) search method to track the direction of a minimum mean-squared error distortion measure. In their method, the distortion for the five initial positions, one at the centre of the co-ordinate and four at co-ordinates $(\pm w/2,\pm w/2)$ of the search window, are computed first. In the next step, three more positions with the same step size in the direction of the previous minimum position are searched. The step size is then halved and the above procedure is continued until the step size becomes unity. Finally all the nine positions are searched. With this method, for $w$=5 pixels/frame, 21 positions are searched as opposed to 121 positions required in the full search method.

Koga *et al.* [8] use a three-step search (TSS) method to compute motion displacements up to 6 pixels/frame. In their method all eight positions surrounding the co-ordinate with a step size of $w/2$ are searched first. At each minimum position the search step size is halved and the next eight new positions are searched. This method, for $w$=6 pixels/frame, searches 25 positions to locate the best match. The technique is the recommended method for the test of software based H.261 [9] for videophone applications.

In Kappagantula and Rao's [4] modified motion estimation algorithm (MMEA), prior to halving the step sizes, two more positions are also searched. With this method for $w$ = 7 pixels/frame, only 19 MAE computations are required. In Srinivasan and Rao's [10] conjugate direction search (CDS) method, at every iteration of the direction search, two conjugate directions with a step size of one pixel, centred at the minimum position, are searched. Thus, for $w$ = 5 pixels/frame, there will be only 13 searches at most.

Another method of fast BMA is the cross-search algorithm (CSA) [11]. In this method, the basic idea is still a logarithmic step search, which has also been exploited in [4,6,8], but with some differences which lead to fewer computational search points. The main difference is that, at each iteration there are four search locations, which are the end points of a cross (×) rather than (+). Also, at the final stage, the search points can be either the end points of (×) or (+) crosses, as shown in Figure 3.7. For a maximum motion displacement of $w$ pixels/frame, the total number of computations becomes $5 + 4 \log_2 w$.

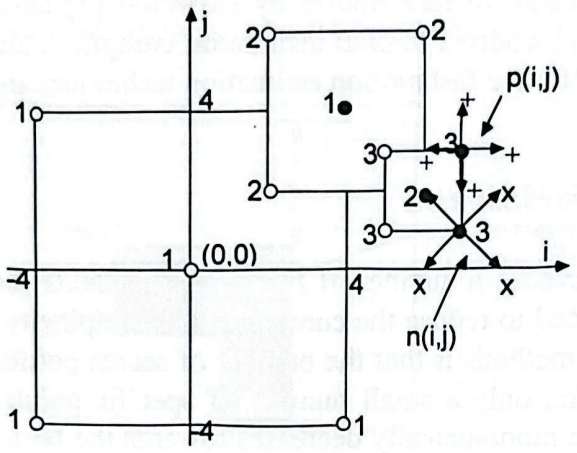



*Figure 3.7. An example of the CSA search for w=8 pixels/frame*

Puri *et al.* [12] have introduced the Orthogonal Search Algorithm (OSA), in which, with a logarithmic step size, at each iteration four new locations are searched. This is the fastest method of all known fast MBAs. In this method, at every step, two positions are searched alternately in the vertical and horizontal directions. The total number of test points is $1 + 4 \log_2 w$.

Table 3.1 shows the computational complexity of various fast search methods, for a range of motion speed from 4 to 16 pixels/frame. The motion compensation efficiency of these algorithms for a motion speed of $w$=8 pixels/frame for two test image sequences are tabulated in Table 3.2.

*Table 3.1 Computational complexity*

| Algorithm | Maximum number of search points | w | | |
|---|---|---|---|---|
| | | 4 | 8 | 16 |
| FSM | $(2w+1)^2$ | 81 | 289 | 1089 |
| TDL | $2 + 7 \log_2 w$ | 16 | 23 | 30 |
| TSS | $1 + 8 \log_2 w$ | 17 | 25 | 33 |
| MMEA | $1 + 6 \log_2 w$ | 13 | 19 | 25 |
| CDS | $3 + 2 w$ | 11 | 19 | 35 |
| OSA | $1 + 4 \log_2 w$ | 9 | 13 | 17 |
| CSA | $5 + 4 \log_2 w$ | 13 | 17 | 21 |

*Table 3.2 Compensation efficiency*

| Algorithm | Split Screen | | Trevor White | |
|---|---|---|---|---|
| | Entropy (bits/pel) | Standard deviation | Entropy (bits/pel) | Standard deviation |
| FSM | 4.57 | 7.39 | 4.41 | 6.07 |
| TDL | 4.74 | 8.23 | 4.60 | 6.92 |
| TSS | 4.74 | 8.19 | 4.58 | 6.86 |
| MMEA | 4.81 | 8.56 | 4.69 | 7.46 |
| CDS | 4.84 | 8.86 | 4.74 | 7.54 |
| OSA | 4.85 | 8.81 | 4.72 | 7.51 |
| CSA | 4.82 | 8.65 | 4.68 | 7.42 |

It can be seen that while fast search methods reduce the computational complexity of the full search method (FSM) significantly, their motion estimation accuracy (compensation efficiency) has not been degraded noticeably.

### *3.3.3 Hierarchical motion estimation*

The assumption of monotonic variation of image intensity, employed in the fast BMAs often causes false estimations, especially for larger picture displacements. These methods perform well for slow moving objects, such as those in video conferencing. However, for higher motion speeds, due to the intrinsic selective nature of these methods, they often converge to a local minimum of distortion.

One method of alleviating this problem is to subsample the image to smaller sizes, such that the motion speed is reduced by the sampling ratio. The process is done on a multilevel image pyramid, known as the *Hierarchical Block Matching Algorithm* (HBMA) [13]. In this technique, pyramids of the image frames are reconstructed, by successive two-dimensional filtering and sub-sampling of the current and past image frames. Figure 3.8 shows a three-level pyramid, where for simplicity, each level of the upper level of the pyramid is taken as the average of four adjacent pixels of one level below. Effectively this is a form of low pass filtering.

Conventional block matching with a block size of 16 pixels, either full search or any fast method, is first applied to the highest level of the pyramid (level 2 in Figure 3.8). This motion vector is then doubled in size, and further refinement within one pixel search is carried out in the following level. The process is repeated to the lowest level. Therefore with an $n$-level pyramid the maximum motion speed of $w$ at the highest level is reduced to $w/2^{n-1}$.

For example a maximum motion speed of 32 pixels/frame with a three-level pyramid is reduced to 8 pixels/frame, which is quite manageable by any fast search method. Note also this method can also be regarded as another type of fast search, with a performance very close to the full search, irrespective of the motion speed, but the computational complexity can be very close to the fast logarithmic methods.

As an example, for a maximum motion speed of 32 pixels/frame, which is very common in high definition video or most TV sports programmes, while the non-hierarchical full search BMA requires $(2\times 32+1)^2 = 4225$ operations, a 4-level hierarchy, where the motion speed at the top level is $32/2^{4-1} = 4$ pixels/frame, only requires $(2\times 4+1)^2 + 3\times 9 = 108$ operations. Here 81 operations are carried out at the top level, and at each lower level nine new positions are searched.

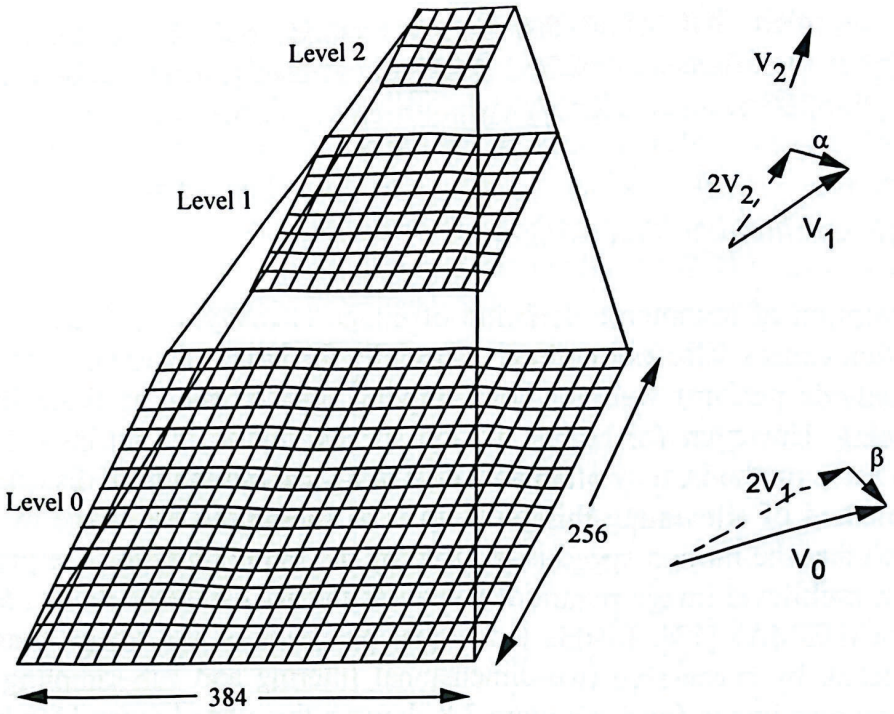



*Figure 3.8 A three-level image pyramid*

## 3.4 Variable length coding

For further bit rate reduction, the transform coefficients and the co-ordinates of the motion vectors are variable length coded (VLC). In VLC, short code words are assigned to the highly probable values and long code words to the less probable ones. The lengths of the codes should vary inversely with the probability of occurrences of the various symbols in VLC. The bit rate required to code these symbols is defined as the logarithm of probability, $p$, at base 2 (bits), i.e. $\log_2 p$. Hence, the entropy of the symbols which is the minimum average bits required to code the symbols can be calculated as

$$H(x) = -\sum_{i=1}^{n} p_i \log_2 p_i \tag{3.10}$$

There are two types of VLC which are employed in the standard video codecs. They are the *Huffman* coding and *Arithmetic* coding. It is noted that Huffman coding is a practical VLC code, but its compression can never reach as low as the entropy due to the constraint that the assigned symbols must have an integral number of bits. However, the arithmetic coding can approach the entropy since the symbols are not coded individually [14]. Huffman coding is employed in all standard codecs to encode the quantised DCT coefficients as well as motion