**Introduction**

The key operation of video compression is to remove redundancy in :

+ Temporal domain.

+ Spatial domain.

+ Frequency domain.

MPEG-4Visual and H.264 (also known as Advanced Video Coding) are standards for the coded

representation of visual information. Each standard is a document that primarily defines two

things, a coded representation (or syntax) that describes visual data in a compressed form and a

method of decoding the syntax to reconstruct visual information.

MPEG-4 Visual and H.264 have related but significantly different visions. Both are concerned

with compression of visual data but MPEG-4 Visual emphasises flexibility whilst

H.264’s emphasis is on efficiency and reliability.

**Video Formats and Quality**

**2.1 INTRODUCTION**

Digital video is a representation of a natural (real-world)

visual scene, sampled spatially and temporally. A scene is sampled at a point in time to

produce a frame (a representation of the complete visual scene at that point in time) or a

field (consisting 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

A typical ‘real world’ or ‘natural’ video scene is composed of multiple objects each with

their own characteristic shape, depth, texture and illumination. The colour and brightness

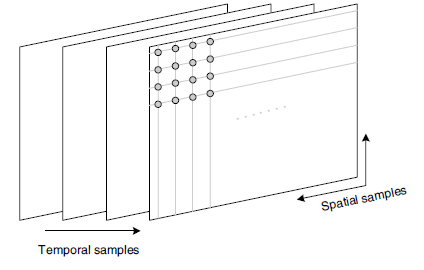
of a natural video scene changes with varying degrees of smoothness throughout the scene

(‘continuous tone’). Characteristics of a typical natural video scene (Figure 2.1) that are

relevant for video processing and compression include spatial characteristics (texture variation

within scene, number and shape of objects, colour, etc.) and temporal characteristics (object

motion, changes in illumination, movement of the camera or viewpoint and so on).



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 (picture element or pixel) is represented

as a number or set of numbers that describes the brightness (luminance) and colour of the

sample.

**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 (picture element or pixel) is represented

as a number or set of numbers that describes the brightness (luminance) and colour of the

sample.

**2.3.1 Spatial Sampling**

The most common format for a sampled image

is a rectangle with the sampling points positioned on a square or rectangular grid.

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 (pixel).The visual

quality of the image is influenced by the number of sampling points.

**2.3.2 Temporal Sampling**

A moving video image is captured by taking a rectangular ‘snapshot’ of the signal at periodic time intervals. Playing back the series of frames produces the appearance of motion. A higher temporal sampling rate (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 are sometimes 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 and 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.

Sampling at 25 or 30 complete frames per second is standard for television pictures (with interlacing to

improve the appearance of motion, see below);

50 or 60 frames per second produces 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 sampled at each temporal sampling interval. A field consists of either the odd-numbered or even-numbered lines within a complete video frame and an interlaced video sequence (Figure 2.6) contains a series of fields, each representing half of the information in a complete video frame (e.g. Figure 2.7 and Figure 2.8). The 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 can appears smoother than in an equivalent progressive video sequence containing 25 frames per second.

**2.4 COLOUR SPACES**

A monochrome image (e.g. Figure 2.1) requires justone number to indicate the brightness or luminance of each spatial sample.

Colour images, onthe other hand, require at least three numbers per pixel position to represent colour accurately.

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).

Any colour can be created by combining red, green and blue in varying proportions

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 Cathode Ray Tubes (CRTs) and Liquid Crystal Displays (LCDs) display 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 YCbCr**

The human visual system (HVS) is less sensitive to colour than to luminance (brightness).

represent a colour image more efficiently by separating the luminance from the colour information and representing luma with a higher resolution than colour.

The YCbCr colour space and its variations (sometimes referred to as YUV) is a popular

way of efficiently representing colour images. Y is the luminance (luma) component and can

be calculated as a weighted average of R, G and B:

*Y* = *kr R* + *kgG* + *kbB*

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* :

*Cb* = *B* − *Y*

*Cr* = *R* − *Y* (2.2)

*Cg* = *G* – *Y*

An RGB image may be converted to YCbCr after capture in order to reduce storage

and/or transmission requirements. Before displaying the image, it is usually necessary to

convert back to RGB.



ITU-R recommendation BT.601 [1] defines *kb* = 0*.*114 and *kr* = 0*.*299. Substituting into the

above equations gives the following widely-used conversion equations:



**2.4.3 YCbCr Sampling Formats**

three sampling patterns for Y, Cb and Cr that are supported by MPEG-4

Visual and H.264

4:4:4 sampling means that the three components (Y, Cb and Cr) have the

same resolution and hence a sample of each component exists at every pixel position. The

numbers indicate the relative sampling rate of each component in the *horizontal* direction. 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 four luminance

samples in the horizontal direction there are two Cb and two Cr samples). 4:2:2 video is used

for high-quality colour reproduction.

In the popular 4:2:0 sampling format (‘YV12’), *Cb* and *Cr* each have half the horizontal

and vertical resolution of *Y*

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 one quarter of the number of samples in the Y component, 4:2:0 YCbCr video

requires exactly half as many samples as 4:4:4 (or R:G:B) video.

4\*4\*3 = 48

4\*4 + 4\*4/2\*2 = 32

4\*4 + 4\*4/2/2\*2 = 24

In a 4:2:0 interlaced video sequence, the Y, Cb and Cr samples corresponding to a

complete video frame are allocated to two fields



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. 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.



Each sample has a possible range of 0 to 255. Levels of 0 and 255 are reserved for synchronisation

and the active luminance signal is restricted to a range of 16 (black) to 235 (white).

**Video Coding Concepts**

**3.1 INTRODUCTION**

Compression is the process of compacting data into a smaller number of bits. Video compression

(video coding) is the process of compacting or condensing a digital video sequence

into a smaller number of bits. ‘Raw’ or uncompressed digital video typically requires a

large bitrate (approximately 216 Mbits for 1 second of uncompressed TV-quality 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)

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 current lossless image compression standards such as JPEG-LS

[1] is a compression ratio of around 3–4 times.

Most video coding methods exploit both *temporal* and *spatial* redundancy to achieve

compression. In the temporal domain, there is usually a high correlation (similarity) between

frames of video that were captured at around the same time.

Temporally adjacent frames (successive

frames in time order) are often highly correlated, especially if the temporal sampling

rate (the 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 and MPEG-4 Visual standards (described in detail in Chapters 5 and 6) share a

number of common features. Both standards assume a CODEC ‘model’ that uses block-based

motion compensation, transform, quantisation and entropy coding. In this chapter we examine

the main components of this model, starting with the temporal model (motion estimation and

compensation) and continuing with image transforms, quantisation, predictive coding and

entropy coding.

**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 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)

tradeoff between bit rate and quality (the rate-distortion trade off)

HVS similar *x* and *y* displacements and so on. Coding efficiency may 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

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 and/or large moving

objects.

The quantisation parameter or quantiser step size controls the tradeoff between compression

efficiency and image quality.

**3.3 TEMPORAL MODEL**

The goal of the temporal model is to reduce redundancy between transmitted frames by forming

a predicted frame and subtracting this from the current frame. The output of this process is

a residual (difference) frame and the more accurate the prediction process, the less energy is

contained in the residual frame.

**3.3.1 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.

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)

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.

**3.3.2 Changes due to Motion**

Changes between video frames may be caused by object motion

It is possible to estimate the trajectory of each pixel between successive

video frames, producing a field of pixel trajectories known as the *optical flow*

**3.3.3 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 *M* × *N* samples in the current frame:

1. Search an area in the reference frame (past or future frame, previously coded and transmitted)

to find a ‘matching’ *M* × *N*-sample region. This is carried out by comparing the

*M* × *N* block in the current frame with some or all of the possible *M* × *N* regions in the

search area (usually a region centred on the current block position) and finding the region

that gives the ‘best’ match.Apopular matching criterion is the energy in the residual formed

by subtracting the candidate region from the current *M* × *N* 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*.

2. The chosen candidate region becomes the predictor for the current *M* × *N* block and is

subtracted from the current block to form a residual *M* × *N* block (*motion compensation*).

3. 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 and decodes the

residual block, adds it to the predictor and reconstructs a version of the original block.

**3.3.4 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

**3.3.5 Motion Compensation Block Size**

smaller motion compensation block sizes can produce better motion compensation results.

However, a smaller block size leads to increased complexity (more search operations must

be carried out) and an increase in the number of motion vectors that need to be transmitted.

**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.

Efficient motion compensation reduces local correlation in the

residual making it easier to compress than the original video frame

The function of the *image model* is to decorrelate image or residual data further 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 (decorrelates and compacts the data), quantisation (reduces

the precision of the transformed data) and reordering (arranges the data to group together

significant values).

**3.4.1 Predictive Image Coding**

Motion compensation is an example of predictive coding

**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 (separated into components with

minimal inter-dependence) and compact (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 (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) [3]. Each of

these operate 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 DiscreteWavelet 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

do not ‘fit’ well with block-based motion compensation. The DCT and the DWT both feature

in MPEG-4 Visual (and a variant of the DCT is 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) and creates **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:

**Y** = **AXA**T (3.1)

and the inverse DCT (IDCT) by:

**X** = **A**T**YA** (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 set of *N* × *N* coefficients representing the image

block data in the DCT domain and these coefficients can be considered as ‘weights’ of a set

of standard *basis patterns*. Any image block may be reconstructed by combining all *N* × *N*

basis patterns, with each basis multiplied by the appropriate weighting factor (coefficient).

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 3.31(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.

***3.4.2.3 Wavelet***

The popular ‘wavelet transform’ (widely used in image compression is based on sets of filters

with coefficients that are equivalent to discrete wavelet functions

A pair of filters are 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 two-dimensional signal such as an intensity

image (Figure 3.32). 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.

Next, each column of these new images is filtered with low- and high-pass filters

(L*y* and H*y* ) and down-sampled by a factor of two to produce four sub-images (LL, LH, HL

and HH).

‘LL’ is the original image, low-pass

filtered in horizontal and vertical directions and subsampled by a factor of 2. ‘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 subband images contain all of the information present in the original image but the sparse nature of the

LH, HL and HH subbands makes them amenable to compression

**3.4.3 Quantisation**

A *scalar quantiser* maps one sample of

the input signal to one quantised output value and a *vector quantiser* maps a group of input

samples (a ‘vector’) to a group of quantised values.

**** where *QP* is a quantisation ‘step size’.

In image and video compression CODECs, the quantisation operation is usually made up

of two parts: a forward quantiser FQ in the encoder and an ‘inverse quantiser’ or (IQ) in the decoder

(in fact quantization is not reversible and so a more accurate term is ‘scaler’ or ‘rescaler’).

If the step size

is large, the range of quantised values is small and can therefore be efficiently represented

(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 quantised values reduces compression efficiency

Quantisation may be used to reduce the precision of image data after applying a transform

such as the DCT or wavelet transform removing remove insignificant values such as near-zero

DCT or wavelet coefficients. The output of a forward quantiser is typically a ‘sparse’ array of quantised

coefficients, mainly containing zeros.

***3.4.3.2 Vector Quantisation***

A vector quantiser 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

1. Partition the original image into regions (e.g. *M* × *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.

**3.4.4 Reordering and Zero Encoding**

Quantised 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 quantiser is a sparse array containing a few nonzero coefficients and a large number of

zero-valued coefficients. Reordering (to group together nonzero coefficients) and efficient

representation of zero coefficients are applied prior to entropy encoding. These processes are

described for the DCT and wavelet transform.

***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. The nonzero DCT coefficients are clustered around the top-left (DC) coefficient and the distribution is roughly symmetrical in the horizontal and vertical directions

For a residual *field* the coefficients are clustered around the DC position but are ‘skewed’, i.e. more nonzero coefficients occur along the left-hand edge of the plot. This is because the field picture has 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

Scan

For a typical frame block with a distribution similar to Figure 3.38, a suitable

scan order is a zigzag starting from the DC (top-left) coefficient



The zig-zag scan may not be ideal for a field block because of the skewed coefficient

distribution (Figure 3.40) and a modified scan order such as Figure 3.42 may be more effective,

in which coefficients on the left-hand side of the block are scanned before those on the righthand

side. 

Run-Level Encoding

The output of the reordering process is an array that typically contains one or more clusters

of nonzero coefficients near the start, followed by strings of zero coefficients. by representing the array as a series of (run, level) pairs where *run* indicates the number

of zeros preceding a nonzero coefficient and *level* indicates the magnitude of the nonzero

coefficient.

Input array: 16,0,0,−3,5,6,0,0,0,0,−7, . . .

Output values: (0,16),(2,−3),(0,5),(0,6),(4,−7). . .

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 quantised to zero and so a reordered

block will usually end in a run of zeros. In so-called ‘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 nonzero 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 nonzero coefficient,

the 3D values are:

(0*,* 16*,* 0)*,* (2*,*−3*,* 0)*,* (0*,* 5*,* 0)*,* (0*,* 6*,* 0)*,* (4*,*−7*,* 1)

**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

quantised transform coefficients (run-level or zerotree encoded as described in Section 3.4.4),

motion vectors (an *x* and *y* displacement vector for each motion-compensated block, with

integer or sub-pixel resolution), . . .

**3.5.1 Predictive Coding**

Certain symbols are highly correlated in local regions of the picture

predicting elements of the current block or macroblock from previously-encoded

data and encoding the difference between the prediction and the actual value

The quantisation parameter or quantiser step size controls the tradeoff between compression

efficiency and image quality. In a real-time video CODEC it may be necessary

to modify the quantisation within an encoded frame

**3.5.2 Variable-length Coding**

A variable-length encoder maps input symbols to a series of codewords (variable length

codes or VLCs).

Frequently-occurring symbols are represented

with short VLCs whilst less common symbols are represented with long VLCs.

***3.5.2.1 Huffman Coding***

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.

***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.Transmitting the information contained

in the probability table to the decoder would add extra overhead and reduce 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, recent image and video coding standards define sets of codewords based

on the probability distributions of ‘generic’ video material

Transform Coefficients (TCOEF)

Motion Vector Difference (MVD)

***3.5.2.3 Other Variable-length Codes***

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 synchronisation and fail to decode

subsequent codes correctly, leading to spreading or propagation of an error in a decoded

sequence. Reversible VLCs (RVLCs) that can be successfully decoded in either a forward or

a backward direction can improve decoding performance when errors occur (see Section 5.3).

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

**3.5.3 Arithmetic Coding**

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.

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Compression of the motion vector field may be improved by predicting each motion

vector from previously-encoded vectors.



Raster scan: a pixel or block is scanned from the top-left of the the first line to the end of the line. Reapeats until all of lines has been scanned.

**6.4.6 Intra Prediction**

For the luma samples, P is

formed for each 4 × 4 block or for a 16 × 16 macroblock. There are a total of nine optional

prediction modes for each 4 × 4 luma block, four modes for a 16 × 16 luma block and four

modes for the chroma components.

The encoder typically selects the prediction mode for each

block that minimises the difference between P and the block to be encoded

***4*** × ***4 Luma Prediction Modes***

The Sum of Absolute Errors (SAE)

for each prediction indicates the magnitude of the prediction error.



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 The samples are interpolated at a 45◦ angle between lower-left

Down-Left) and upper-right.

Mode 4 (Diagonal The samples are extrapolated at a 45◦ angle down and to the right.

Down-Right)

Mode 5 (Vertical-Right) Extrapolation at an angle of approximately 26.6◦ to the left of

vertical (width/height = 1*/*2).

Mode 6 (Horizontal-Down) Extrapolation at an angle of approximately 26.6◦ below

horizontal.

Mode 7 (Vertical-Left) 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.

***6.4.6.2 16*** × ***16 Luma Prediction Modes***

Mode 0 (vertical) Extrapolation from upper samples (H)

Mode 1 (horizontal) Extrapolation from left samples (V)

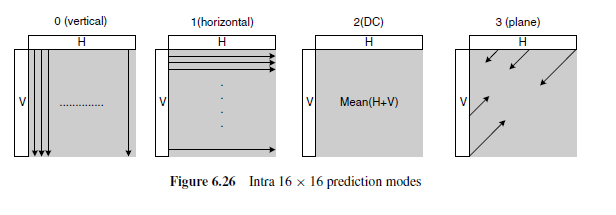
Mode 2 (DC) Mean of upper and left-hand samples (H + V).

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.

Intra 16 × 16 mode works best in homogeneous areas of

an image.



***6.4.6.3 8*** × ***8 Chroma Prediction Modes***

The four prediction modes are very similar to the 16×16 luma

prediction modes described in Section 6.4.6.2 and illustrated in Figure 6.26, except that the

numbering of the modes is different. The modes areDC(mode 0), horizontal (mode 1), vertical

(mode 2) and plane (mode 3).

***6.4.6.4 Signalling Intra Prediction Modes***

The choice of intra prediction mode for each 4 × 4 block must be signalled to the decoder and

this could potentially require a large number of bits

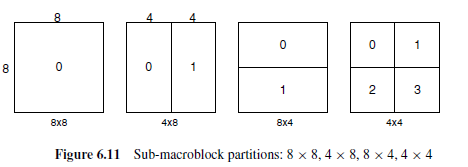
**6.4.5 Inter Prediction**

***6.4.5.1 Tree structured motion compensation***

The luminance component of each macroblock (16×16 samples) may be split up in four ways



If the 8×8 mode is chosen, each of the four 8×8 sub-macroblocks within the macroblock may be split in a further 4 ways



These partitions and

sub-macroblock give rise to a large number of possible combinations within each macroblock.

This method of partitioning macroblocks into motion compensated sub-blocks of varying size

is known as *tree structured motion compensation*.

In general, a large partition size is

appropriate for homogeneous areas of the frame and a small partition size may be beneficial

for detailed areas.

***6.4.5.2 Motion Vectors***

Each partition or sub-macroblock 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 quarter-sample resolution for the luma component and one-eighth-sample resolution for

the chroma components.

Generating Interpolated Samples

interpolated

from integer-position samples using a six 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

six horizontal integer samples E, F, G, H, I and J:

**b** = round((E − 5F + 20G + 20H − 5I + J) */*32)

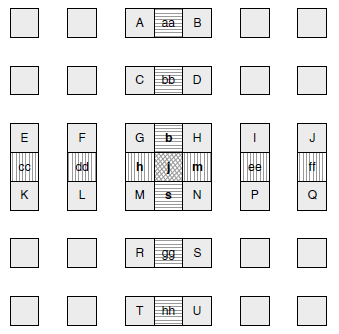
the remaining half-pel

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

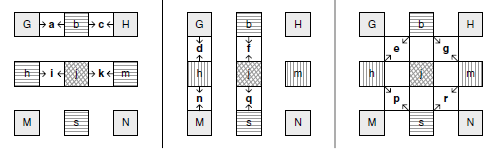
(note that the result is the same whether **j** is interpolated horizontally or vertically; note also

that un-rounded versions of h and m are used to generate j).

 ****

Once all the half-pel samples are available, the samples at quarter-step (‘quarter-pel’)

positions are produced by linear interpolation

Quarter-pel positions with two

horizontally or vertically adjacent half- or integer-position samples are linearly interpolated between these adjacent samples, for example:

**a** = round((G + b) */* 2)

The remaining quarter-pel positions (**e**, **g**, **p** and **r** in the figure) are linearly interpolated between

a pair of diagonally opposite *half* -pel samples. For example, **e** is interpolated between b and

h.

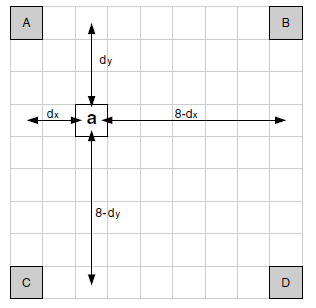
Quarter-pel resolution motion vectors in the luma component require eighth-sample

resolution vectors in the chroma components (assuming 4:2:0 sampling).

generated at eighth-sample intervals between integer samples in each chroma component

using linear interpolation

**a** = round([(8 − d*x* ) · (8 − d*y*)A + d*x* · (8 − d*y*)B + (8 − d*x* ) · d*y*C + d*x* · d*y*D]*/*64)

***6.4.5.3 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

transmitted. The method of forming the prediction MVp depends on the motion compensation

partition size and on the availability of nearby vectors.

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.

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 (i.e. as

if the block were encoded in 16 × 16 Inter mode).

**6.4.7 Deblocking Filter**

A filter is applied to each decoded macroblock to reduce blocking distortion. 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

Filtering is applied to vertical or horizontal edges of 4×4 blocks in a macroblock (except

for 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 6.29).

2. Filter 4 horizontal boundaries of the luma component (in order e, f, g, h, Figure 6.29).

3. Filter 2 vertical boundaries of each chroma component (i, j).

4. Filter 2 horizontal boundaries of each chroma component (k, l).

ToC

**Definitions**

**Capturing Video**

**Spatial Sampling**

**Temporal Sampling**

**Frames and Fields**

**Color Spaces**

**RBG**

**YCbCr (YUV)**

**The Relation between RBG and YUV**

**YCbCr Sampling Formats**

**Video Coding Concepts**

**Video Codec**

**Video Encoder**

**Video Decoder**

**Temporal Model**

**Block-based Motion Estimation and Compensation**

**Macroblock & Block Size**

**Spatial Model**

**Transform Coding**

**The Discrete Cosine Transform (DCT)**

**Quantisation**

**Reordering and Zero Encoding**

**Entropy Encoding**

**Predictive Coding**

**Variable-length Coding**

**Arithmetic Coding**

**Arithmetic Encoding Example**

**Arithmetic Decoding Example**

**H.264/MPEG4 Part 10**

**Intra Prediction**

***4*** × ***4 Luma Prediction Modes***

***16*** × ***16 Luma Prediction Modes***

***8*** × ***8 Chroma Prediction Modes***

**Inter Prediction**

***Tree structured motion compensation***

***Motion Vectors & Motion Vector Prediction***

**Deblocking Filter**