THE H.264 ADVANCED VIDEO COMPRESSION STANDARD

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

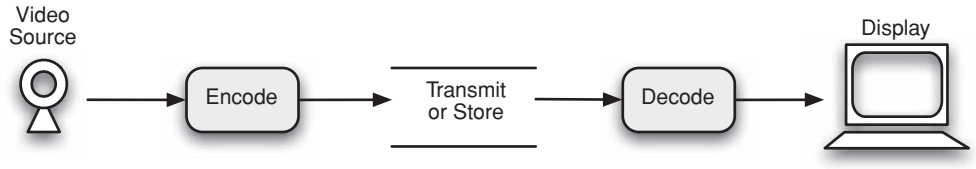

Figure 3.1 Encoder / Decoder

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

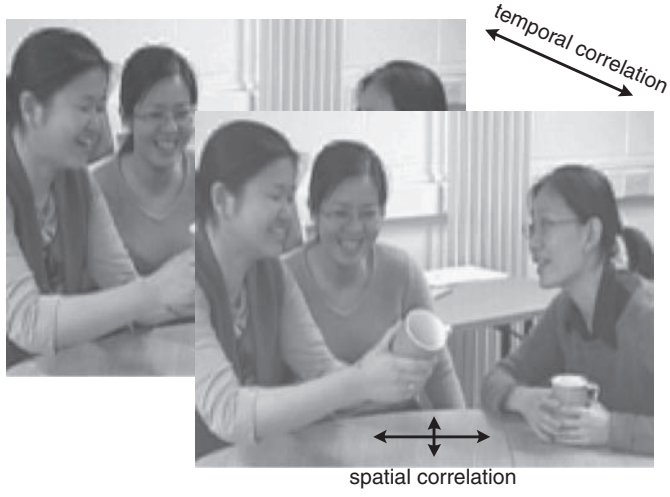

Figure 3.2 Spatial and temporal correlation in a video sequence

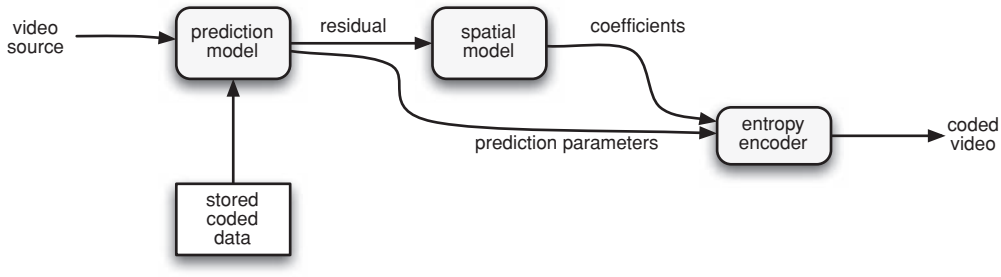

Figure 3.3 Video encoder block diagram

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

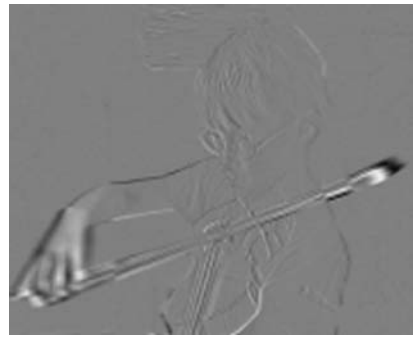

Figure 3.6 Difference

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,

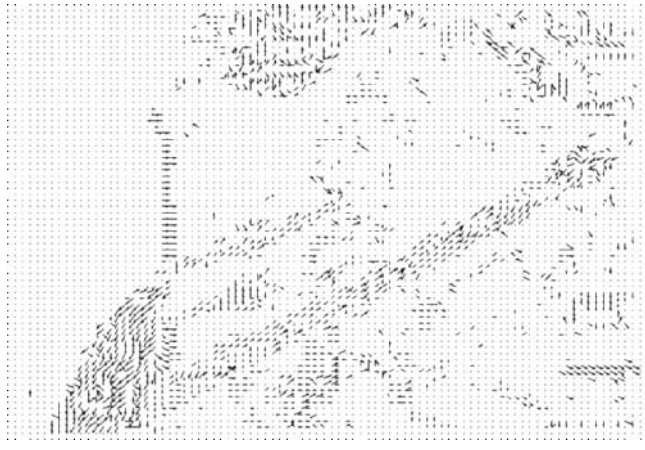

Figure 3.7 Optical flow

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

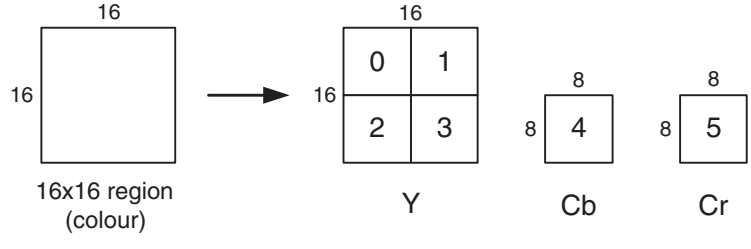

Figure 3.8 Macroblock (4:2:0)

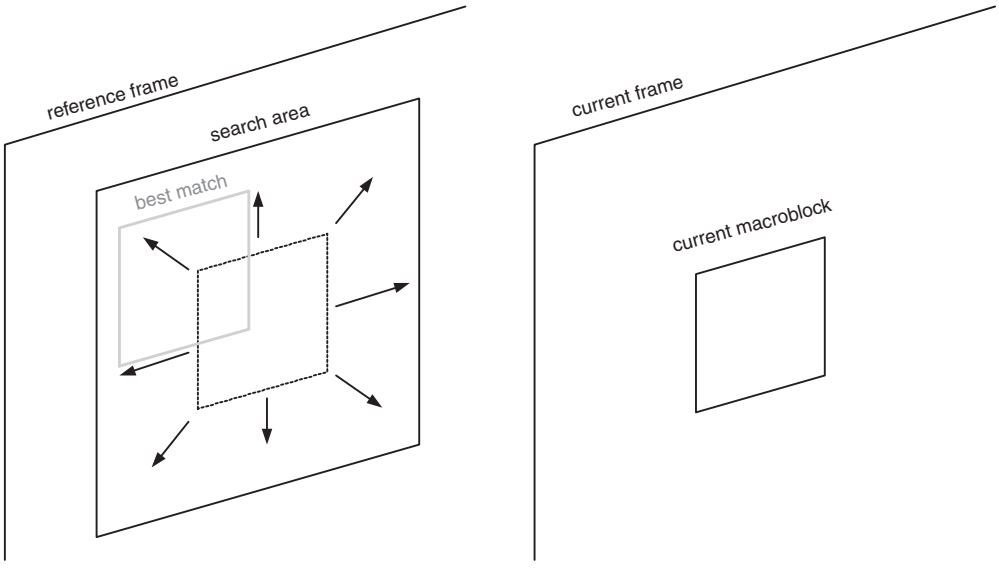

Figure 3.9 Motion estimation

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

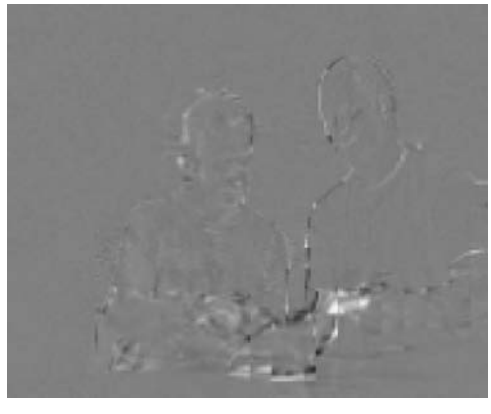

Figure 3.14 Residual : 8 × 8 block size

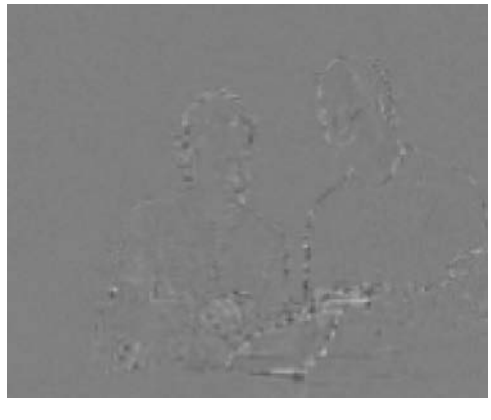

Figure 3.15 Residual : 4 × 4 block size

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

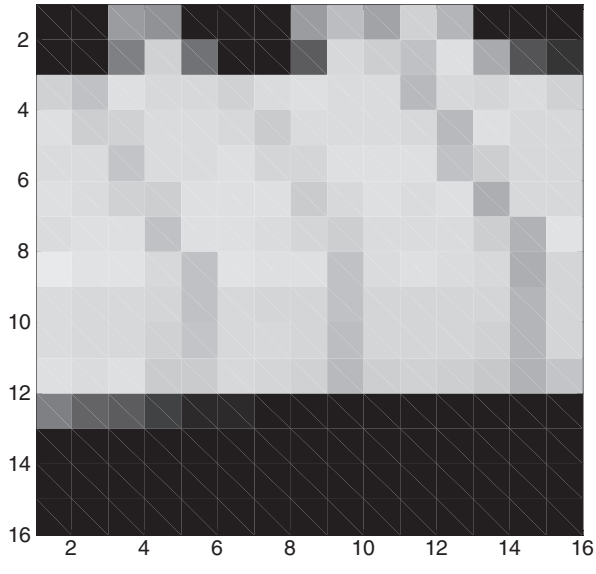

Figure 3.16 Close-up of reference region

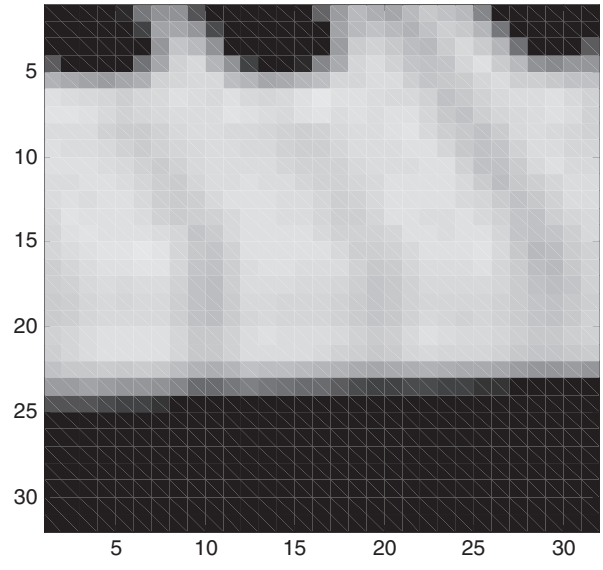

Figure 3.17 Reference region interpolated to half-pixel positions

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

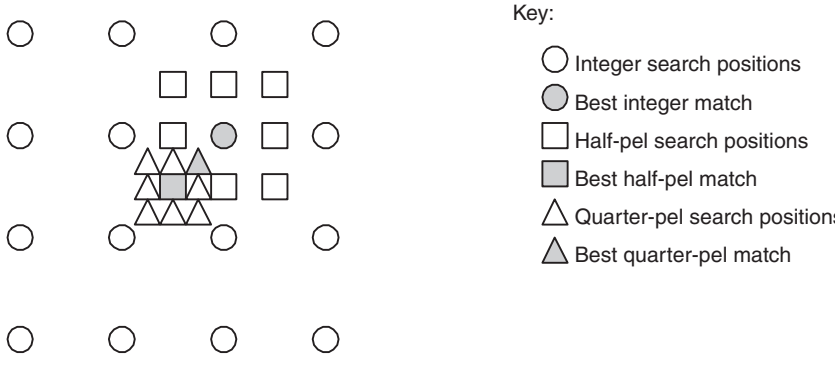

Figure 3.18 Integer, half-pixel and quarter-pixel motion estimation

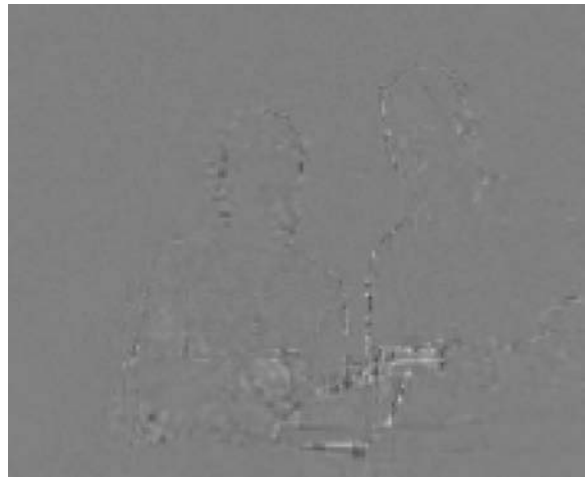

Figure 3.19 Residual : 4 × 4 blocks, 1/2-pixel compensation

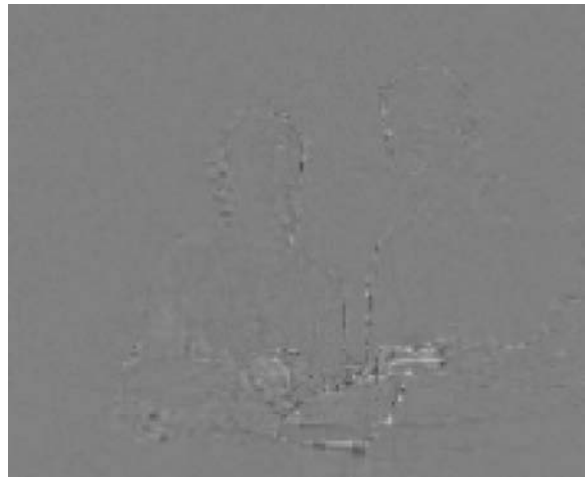

Figure 3.20 Residual : 4 × 4 blocks, 1/4-pixel compensation

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 ¼-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 ½ or ¼-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 ¼-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

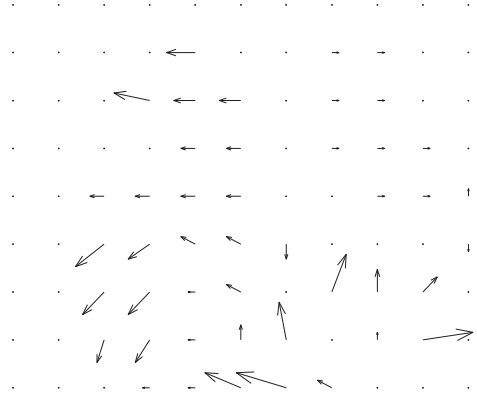

Figure 3.21 Motion vector map : 16 × 16 blocks, integer vectors

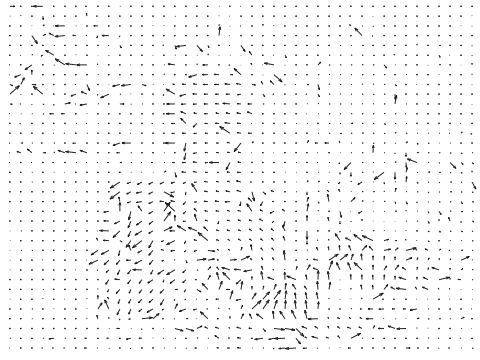

Figure 3.22 Motion vector map : 4 × 4 blocks, 1/4-pixel vectors

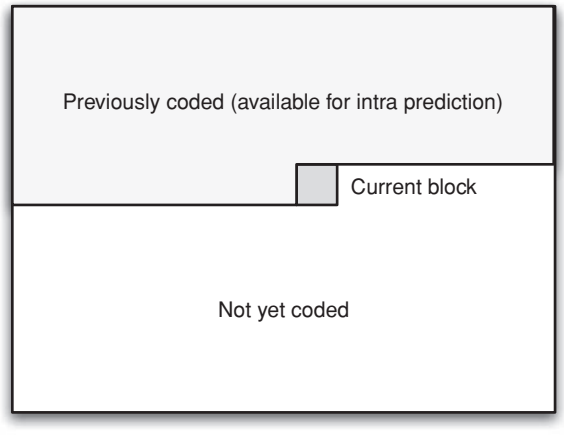

Figure 3.23 Intra prediction: available samples

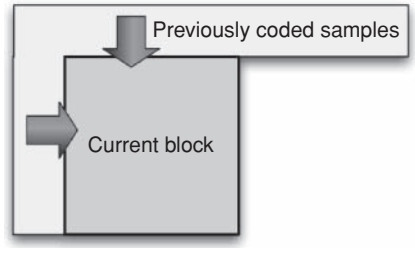

Figure 3.24 Intra prediction: spatial extrapolation

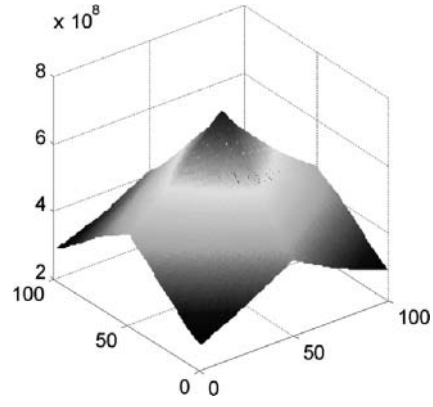

Figure 3.25 2D autocorrelation function of image

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.

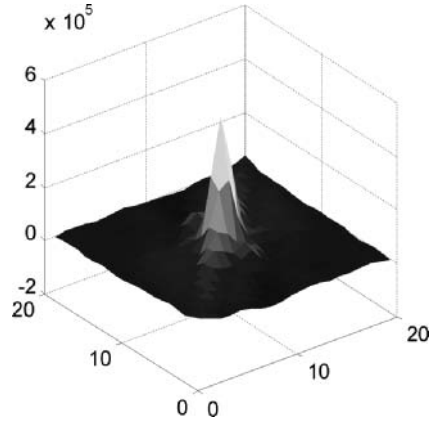

Figure 3.26 2D autocorrelation function of residual

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

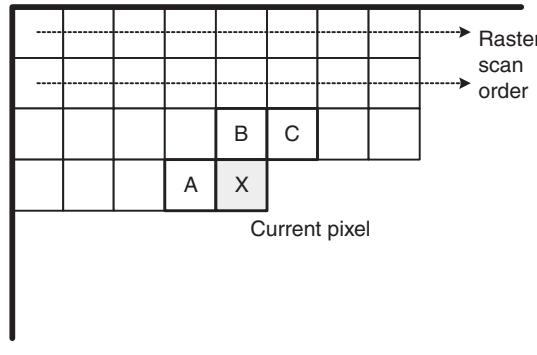

Figure 3.27 Spatial prediction (DPCM)

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:

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

(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).

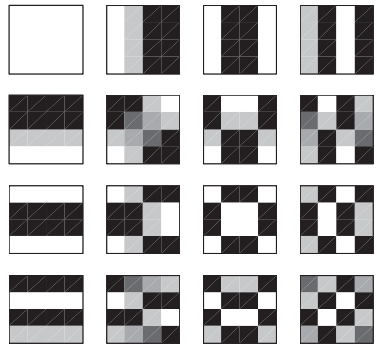

Figure 3.28 4 × 4 DCT basis patterns

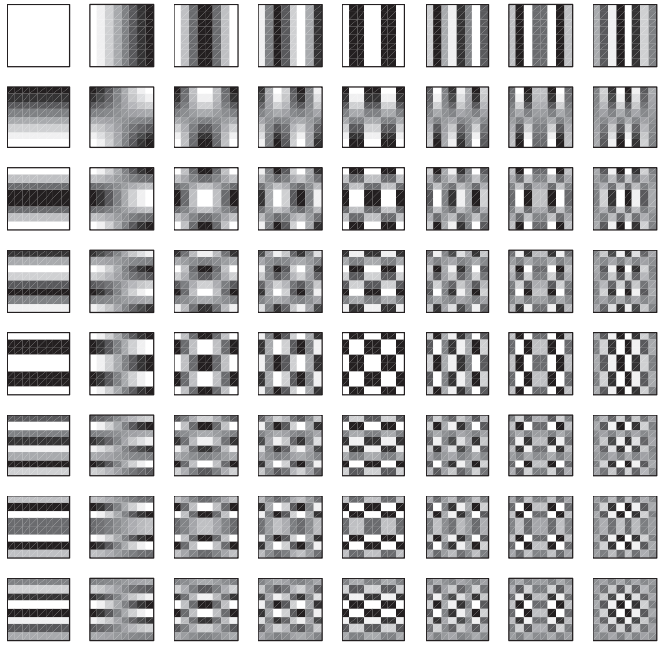

Figure 3.29 8 × 8 DCT basis patterns

Example 1 Calculating the DCT of a 4 × 4 block

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

	J = 0	1	2	3
i= 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, $\mathbf{Y} = \mathbf{Y'A^T}$, 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;

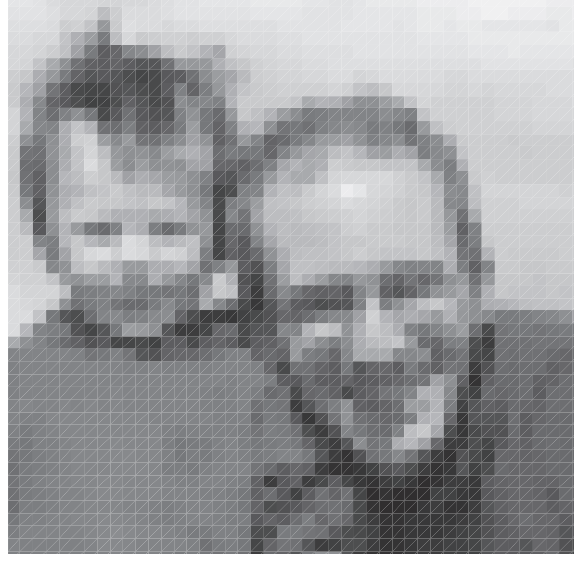

Figure 3.30 Image section showing 4 × 4 block

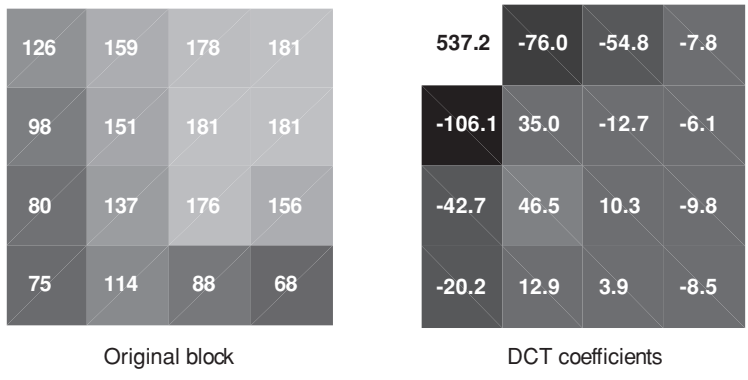

Figure 3.31 Close-up of 4 × 4 block; DCT coefficients

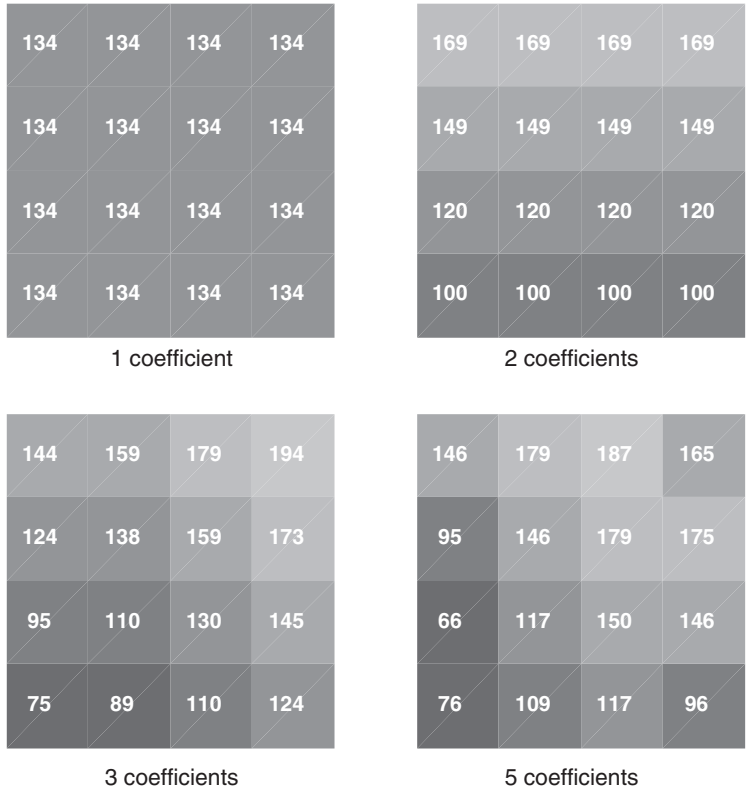

Figure 3.32 Block reconstructed from (a) 1, (b) 2, (c) 3, (d) 5 coefficients

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

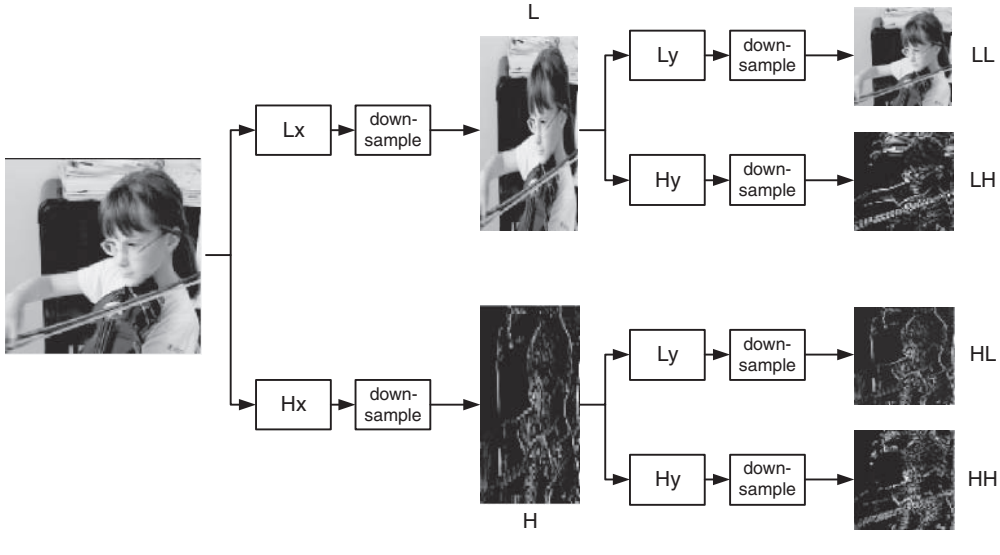

Figure 3.33 Two-dimensional wavelet decomposition process

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

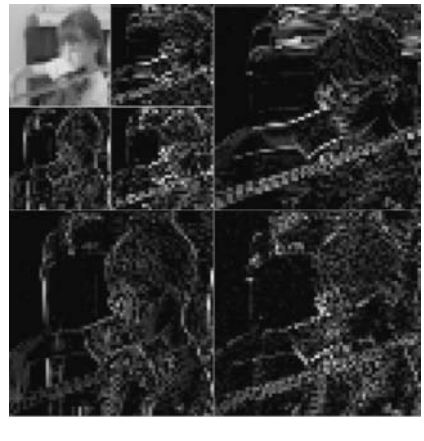

Figure 3.35 Two-stage wavelet decomposition of image

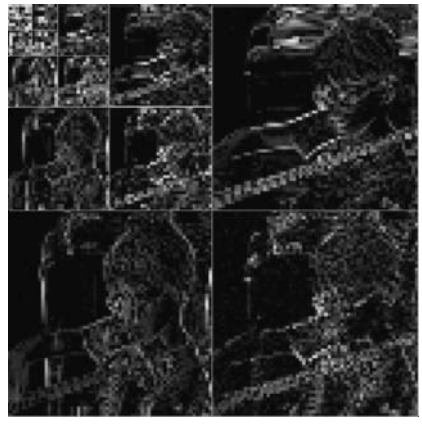

Figure 3.36 Five-stage wavelet decomposition of image

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.

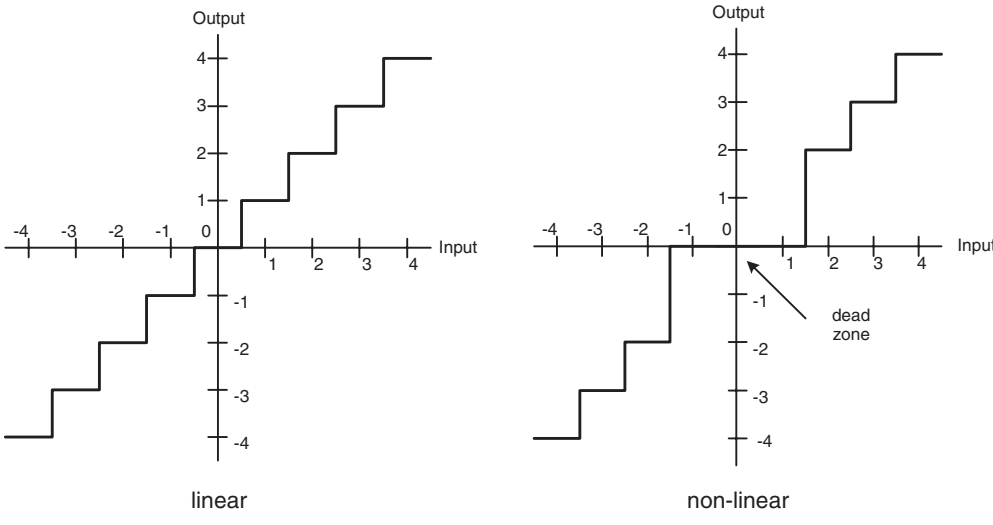

Figure 3.37 Scalar quantizers: linear; non-linear with dead zone

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.

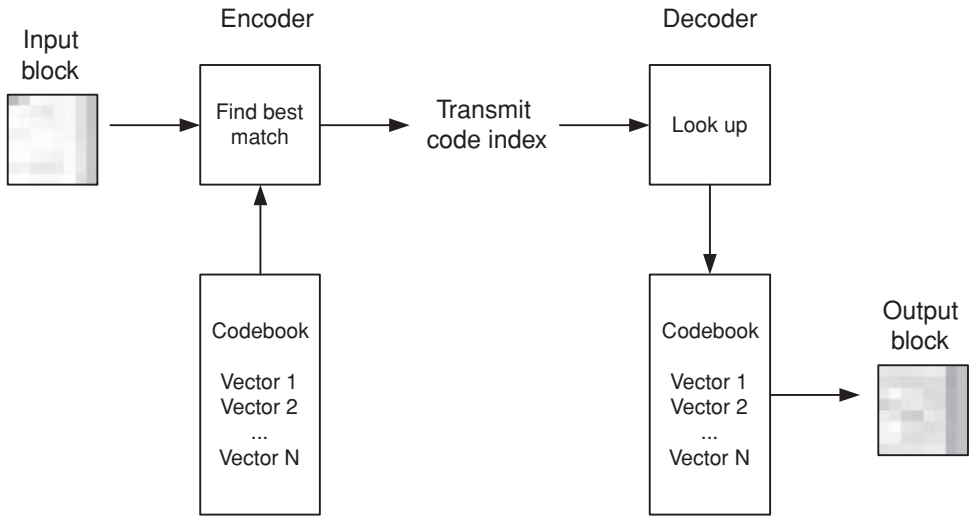

Figure 3.38 Vector quantization

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.

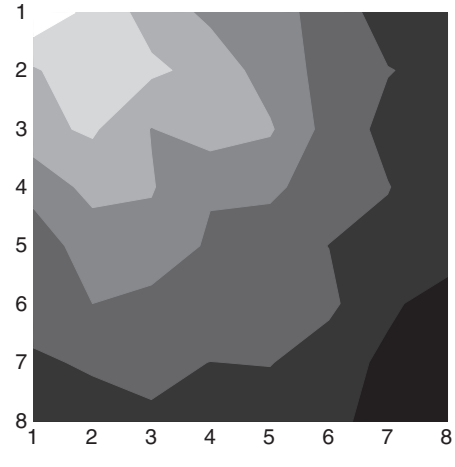

Figure 3.39 8 × 8 DCT coefficient distribution (frame)

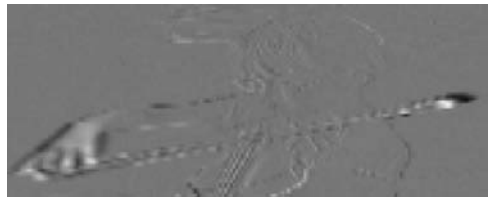

Figure 3.40 Residual field picture

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.

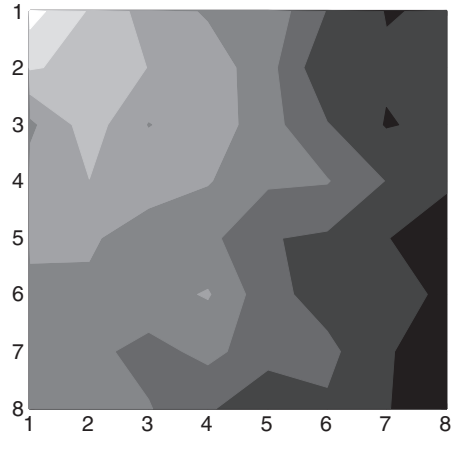

Figure 3.41 8 × 8 DCT coefficient distribution (field)

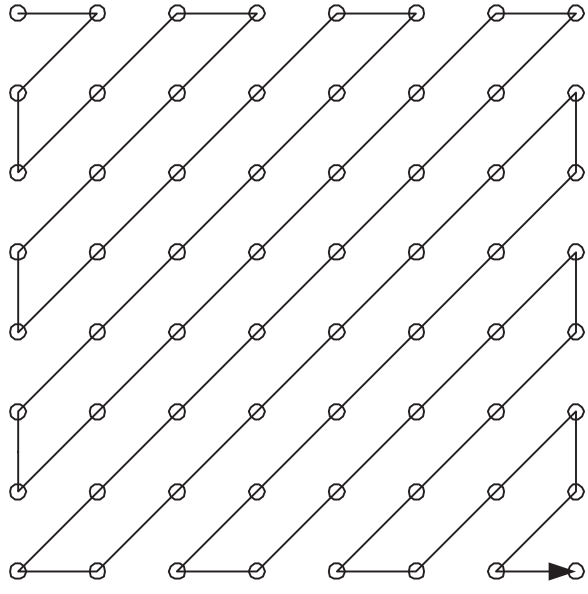

Figure 3.42 Zigzag scan example : frame block

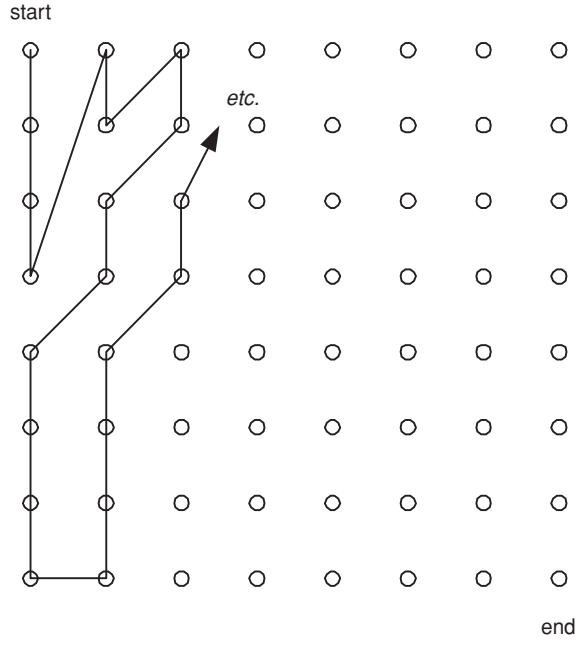

Figure 3.43 Zigzag scan example : field block

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

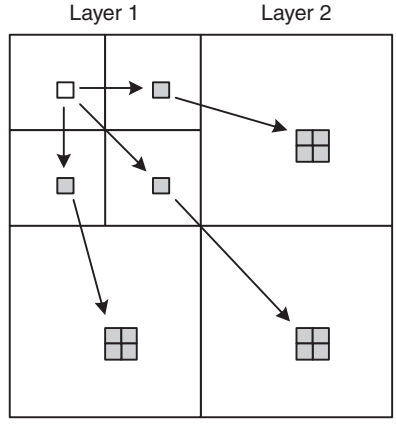

Figure 3.44 Wavelet coefficient and 'children'

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.

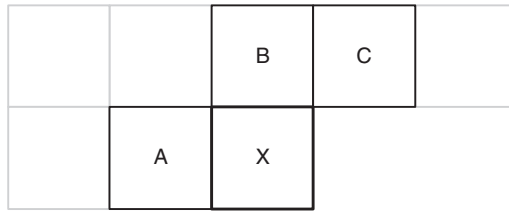

Figure 3.45 Motion vector prediction candidates

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.

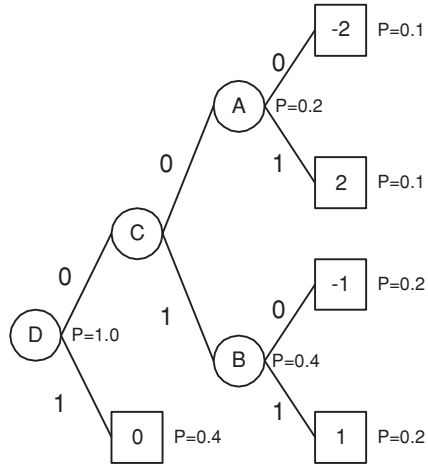

Figure 3.46 Generating the Huffman code tree: Sequence 1 motion vectors

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.

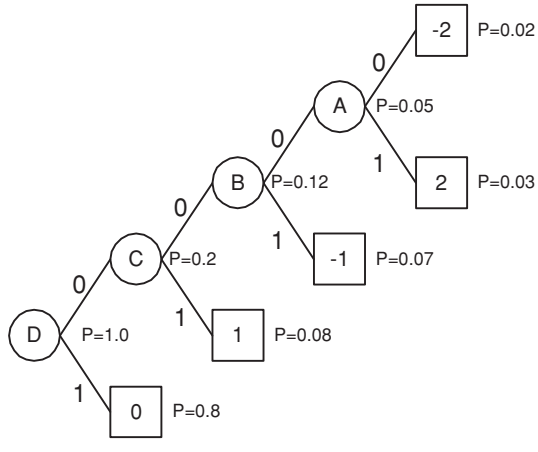

Figure 3.47 Huffman tree for sequence 2 motion vectors

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.

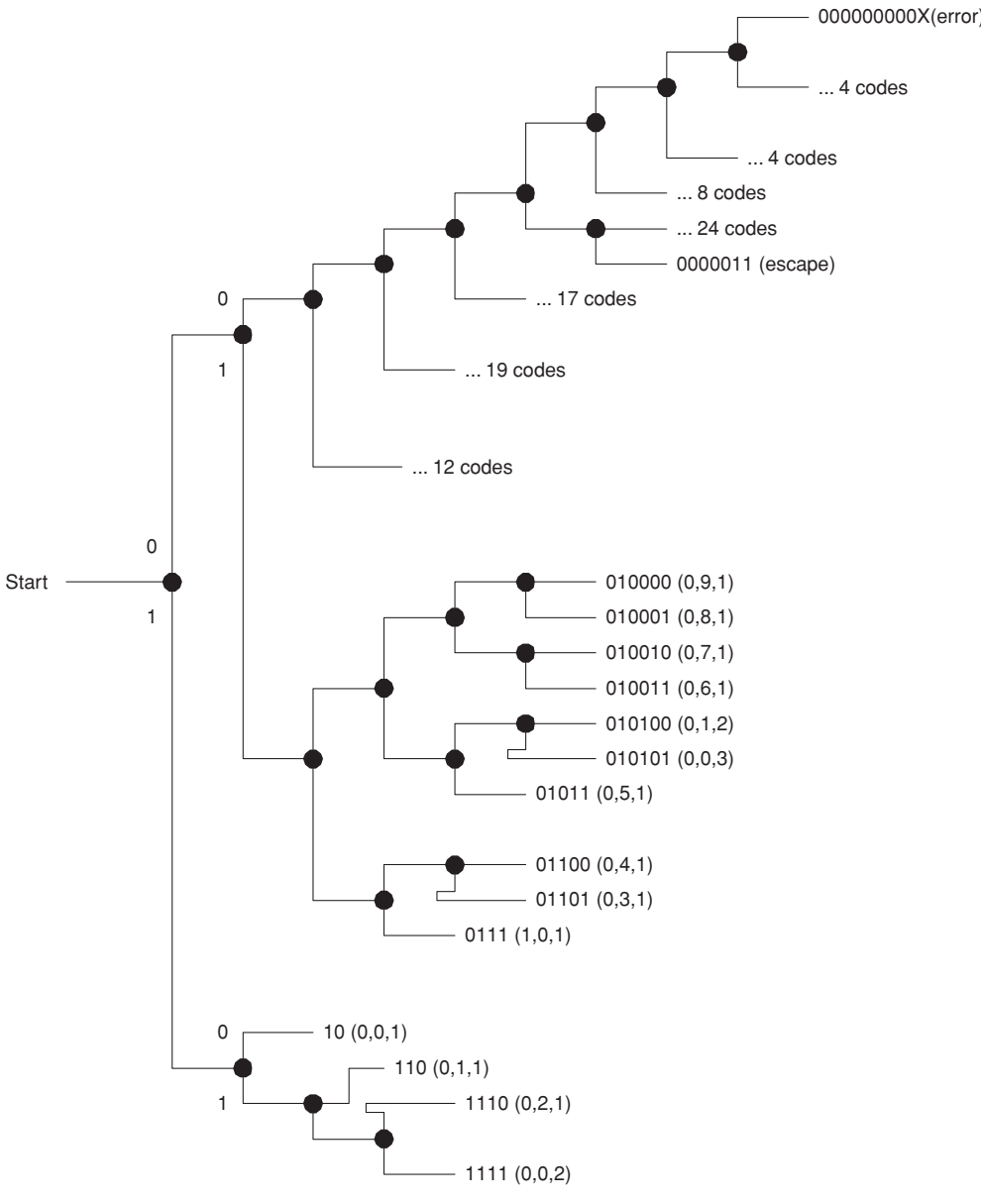

Figure 3.48 MPEG4 TCOEF VLCs (partial)

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

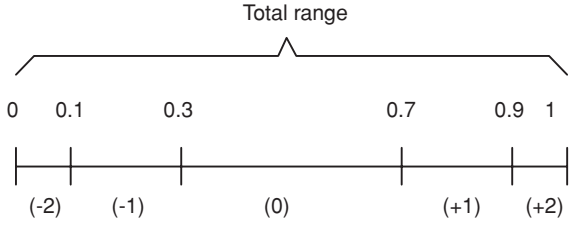

Figure 3.49 Sub-range example

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

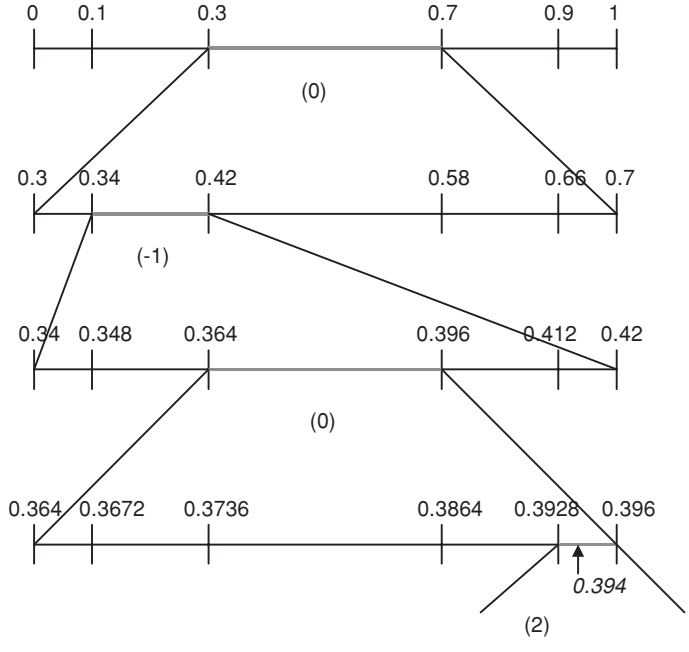

Figure 3.50 Arithmetic coding example

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.

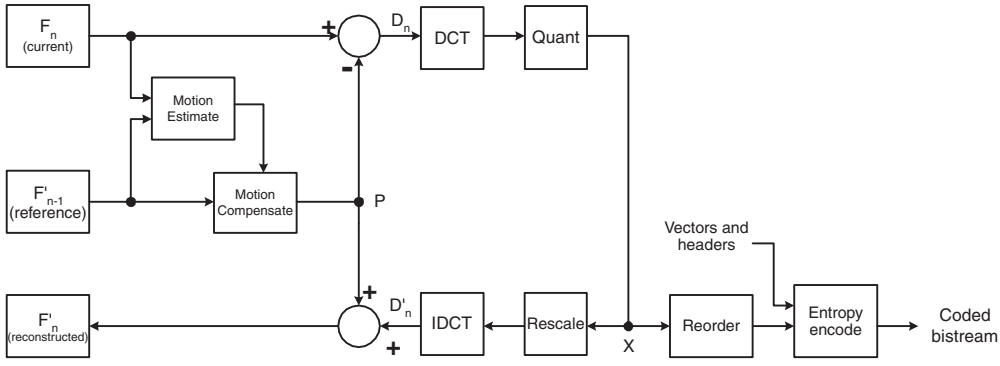

Figure 3.51 DPCM/DCT video encoder

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.

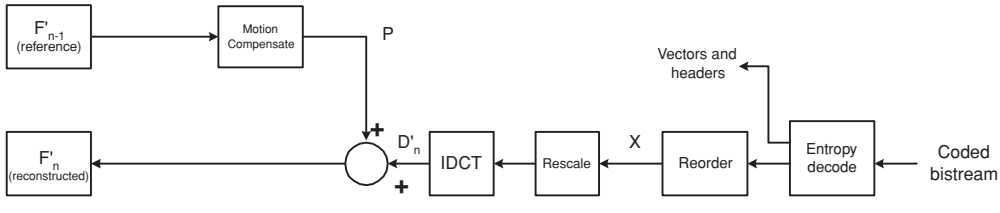

Figure 3.52 DPCM/DCT video decoder

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

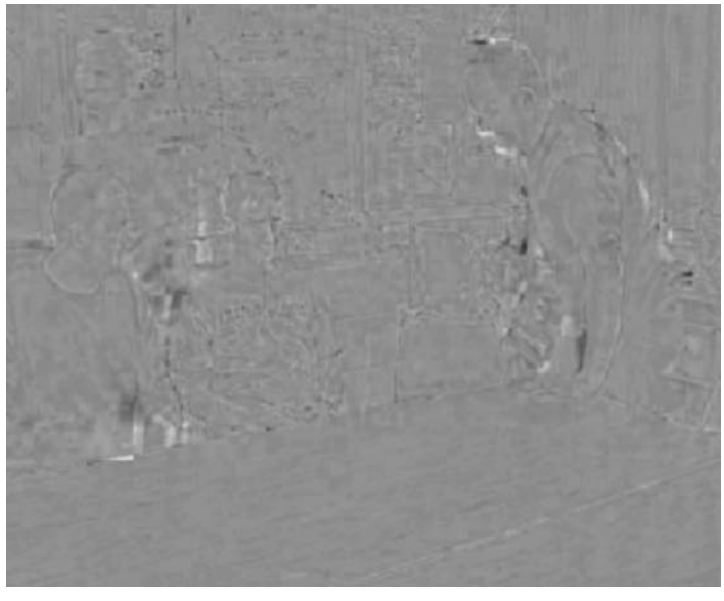

Figure 3.58 Motion compensated residual frame

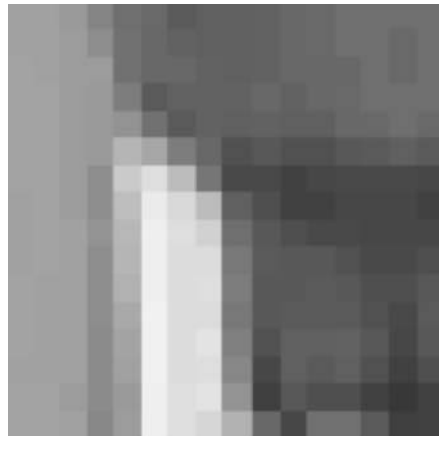

Figure 3.59 Original macroblock : luminance

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

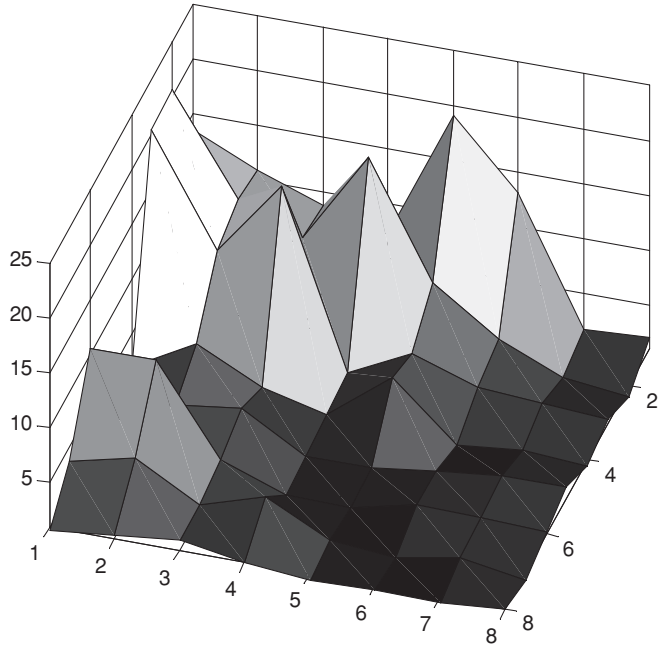

Figure 3.61 DCT coefficient magnitudes : top-right 8 × 8 block

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.

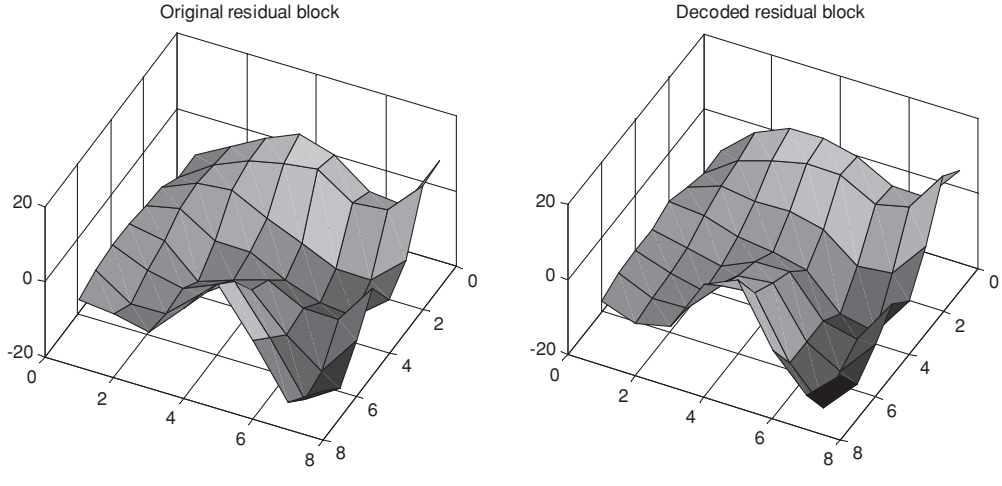

Figure 3.62 Comparison of original and decoded residual blocks

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').