6

H.264/MPEG4 Part 10

6.1 INTRODUCTION

The Moving Picture Experts Group and the Video Coding Experts Group (MPEG and VCEG) have developed a new standard that promises to outperform the earlier MPEG-4 and H.263 standards, providing better compression of video images. The new standard is entitled 'Advanced Video Coding' (AVC) and is published jointly as Part 10 of MPEG-4 and ITU-T Recommendation H.264 [1, 3].

6.1.1 Terminology

Some of the important terminology adopted in the H.264 standard is as follows (the details of these concepts are explained in later sections):

A field (of interlaced video) or a frame (of progressive or interlaced video) is encoded to produce a *coded picture*. A coded frame has a *frame number* (signalled in the bitstream), which is not necessarily related to decoding order, and each coded field of a progressive or interlaced frame has an associated *picture order count*, which defines the decoding order of fields. Previously coded pictures (*reference pictures*) may be used for inter prediction of further coded pictures. Reference pictures are organised into one or two lists (sets of numbers corresponding to reference pictures), described as *list 0* and *list 1*.

A coded picture consists of a number of *macroblocks*, each containing 16 × 16 luma samples and associated chroma samples (8 × 8 Cb and 8 × 8 Cr samples in the current standard). Within each picture, macroblocks are arranged in *slices*, where a slice is a set of macroblocks in raster scan order (but not necessarily contiguous – see section 6.4.3). An *I slice* may contain only I macroblock types (see below), a *P slice* may contain P and I macroblock types and a *B slice* may contain B and I macroblock types. (There are two further slice types, SI and SP, discussed in section 6.6.1).

I macroblocks are predicted using intra prediction from decoded samples in the current slice. A prediction is formed either (a) for the complete macroblock or (b) for each 4 × 4 block

H.264 and MPEG-4 Video Compression: Video Coding for Next-generation Multimedia.
Iain E. G. Richardson. ISBN: 0-470-84837-5

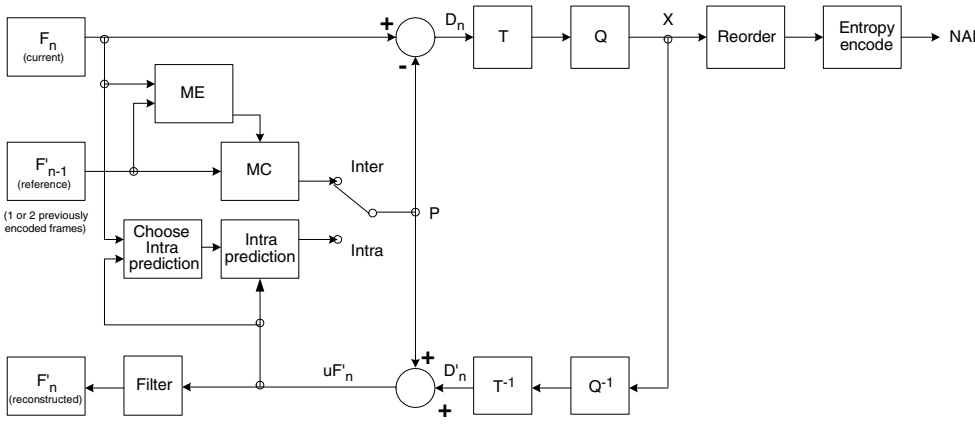


Figure 6.1 H.264 Encoder

of luma samples (and associated chroma samples) in the macroblock. (An alternative to intra prediction, I_PCM mode, is described in section 6.4.6).

P macroblocks are predicted using inter prediction from reference picture(s). An inter coded macroblock may be divided into *macroblock partitions*, i.e. blocks of size 16 × 16, 16 × 8, 8 × 16 or 8 × 8 luma samples (and associated chroma samples). If the 8 × 8 partition size is chosen, each 8 × 8 *sub-macroblock* may be further divided into *sub-macroblock partitions* of size 8 × 8, 8 × 4, 4 × 8 or 4 × 4 luma samples (and associated chroma samples). Each macroblock partition may be predicted from one picture in list 0. If present, every sub-macroblock partition in a sub-macroblock is predicted from the same picture in list 0.

B macroblocks are predicted using inter prediction from reference picture(s). Each macroblock partition may be predicted from one or two reference pictures, one picture in list 0 and/or one picture in list 1. If present, every sub-macroblock partition in a sub-macroblock is predicted from (the same) one or two reference pictures, one picture in list 0 and/or one picture in list 1.

6.2 THE H.264 CODEC

In common with earlier coding standards, H.264 does not explicitly define a CODEC (enCOder / DECoder pair) but rather defines the syntax of an encoded video bitstream together with the method of decoding this bitstream. In practice, a compliant encoder and decoder are likely to include the functional elements shown in Figure 6.1 and Figure 6.2. With the exception of the deblocking filter, most of the basic functional elements (prediction, transform, quantisation, entropy encoding) are present in previous standards (MPEG-1, MPEG-2, MPEG-4, H.261, H.263) but the important changes in H.264 occur in the details of each functional block.

The Encoder (Figure 6.1) includes two dataflow paths, a 'forward' path (left to right) and a 'reconstruction' path (right to left). The dataflow path in the Decoder (Figure 6.2) is shown from right to left to illustrate the similarities between Encoder and Decoder. Before examining the detail of H.264, we will describe the main steps in encoding and decoding a frame (or field)

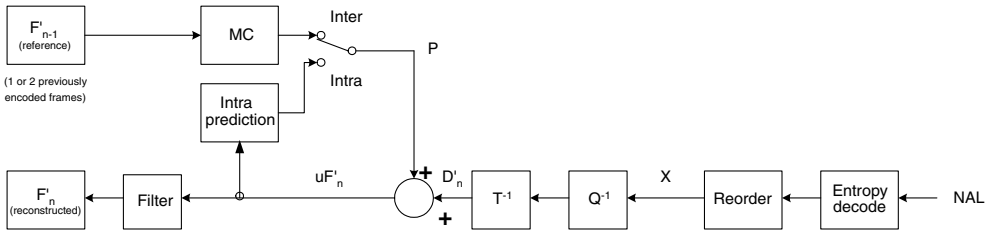


Figure 6.2 H.264 Decoder

of video. The following description is simplified in order to provide an overview of encoding and decoding. The term "block" is used to denote a macroblock partition or sub-macroblock partition (inter coding) or a 16 × 16 or 4× 4 block of luma samples and associated chroma samples (intra coding).

Encoder (forward Path)

An input frame or field F_n is processed in units of a macroblock. Each macroblock is encoded in intra or inter mode and, for each block in the macroblock, a prediction PRED (marked 'P' in Figure 6.1) is formed based on reconstructed picture samples. In Intra mode, PRED is formed from samples in the current slice that have previously encoded, decoded and reconstructed (uF'_n in the figures; note that *unfiltered* samples are used to form PRED). In Inter mode, PRED is formed by motion-compensated prediction from one or two reference picture(s) selected from the set of list 0 and/or list 1 reference pictures. In the figures, the reference picture is shown as the previous encoded picture F'_{n-1} but the prediction reference for each macroblock partition (in inter mode) may be chosen from a selection of past or future pictures (in display order) that have already been encoded, reconstructed and filtered.

The prediction PRED is subtracted from the current block to produce a residual (difference) block D_n that is transformed (using a block transform) and quantised to give X, a set of quantised transform coefficients which are reordered and entropy encoded. The entropy-encoded coefficients, together with side information required to decode each block within the macroblock (prediction modes, quantiser parameter, motion vector information, etc.) form the compressed bitstream which is passed to a Network Abstraction Layer (NAL) for transmission or storage.

Encoder (Reconstruction Path)

As well as encoding and transmitting each block in a macroblock, the encoder decodes (reconstructs) it to provide a reference for further predictions. The coefficients X are scaled (Q^{-1}) and inverse transformed (T^{-1}) to produce a difference block D'_n. The prediction block PRED is added to D'_n to create a reconstructed block uF'_n (a decoded version of the original block; u indicates that it is unfiltered). A filter is applied to reduce the effects of blocking distortion and the reconstructed reference picture is created from a series of blocks F'_n.

Decoder

The decoder receives a compressed bitstream from the NAL and entropy decodes the data elements to produce a set of quantised coefficients X. These are scaled and inverse transformed

to give D'_n (identical to the D'_n shown in the Encoder). Using the header information decoded from the bitstream, the decoder creates a prediction block PRED, identical to the original prediction PRED formed in the encoder. PRED is added to D'_n to produce uF'_n which is filtered to create each decoded block F'_n.

6.3 H.264 STRUCTURE

6.3.1 Profiles and Levels

H.264 defines a set of three *Profiles*, each supporting a particular set of coding functions and each specifying what is required of an encoder or decoder that complies with the Profile. The *Baseline Profile* supports intra and inter-coding (using I-slices and P-slices) and entropy coding with context-adaptive variable-length codes (CAVLC). The *Main Profile* includes support for interlaced video, inter-coding using B-slices, inter coding using weighted prediction and entropy coding using context-based arithmetic coding (CABAC). The *Extended Profile* does not support interlaced video or CABAC but adds modes to enable efficient switching between coded bitstreams (SP- and SI-slices) and improved error resilience (Data Partitioning). Potential applications of the Baseline Profile include videotelephony, videoconferencing and wireless communications; potential applications of the Main Profile include television broadcasting and video storage; and the Extended Profile may be particularly useful for streaming media applications. However, each Profile has sufficient flexibility to support a wide range of applications and so these examples of applications should not be considered definitive.

Figure 6.3 shows the relationship between the three Profiles and the coding tools supported by the standard. It is clear from this figure that the Baseline Profile is a subset of the Extended Profile, but not of the Main Profile. The details of each coding tool are described in Sections 6.4, 6.5 and 6.6 (starting with the Baseline Profile tools).

Performance limits for CODECs are defined by a set of Levels, each placing limits on parameters such as sample processing rate, picture size, coded bitrate and memory requirements.

6.3.2 Video Format

H.264 supports coding and decoding of 4:2:0 progressive or interlaced video[1] and the default sampling format for 4:2:0 progressive frames is shown in Figure 2.11 (other sampling formats may be signalled as Video Usability Information parameters). In the default sampling format, chroma (Cb and Cr) samples are aligned horizontally with every 2nd luma sample and are located vertically between two luma samples. An interlaced frame consists of two fields (a top field and a bottom field) separated in time and with the default sampling format shown in Figure 2.12.

[1] An extension to H.264 to support alternative colour sampling structures and higher sample accuracy is currently under development.

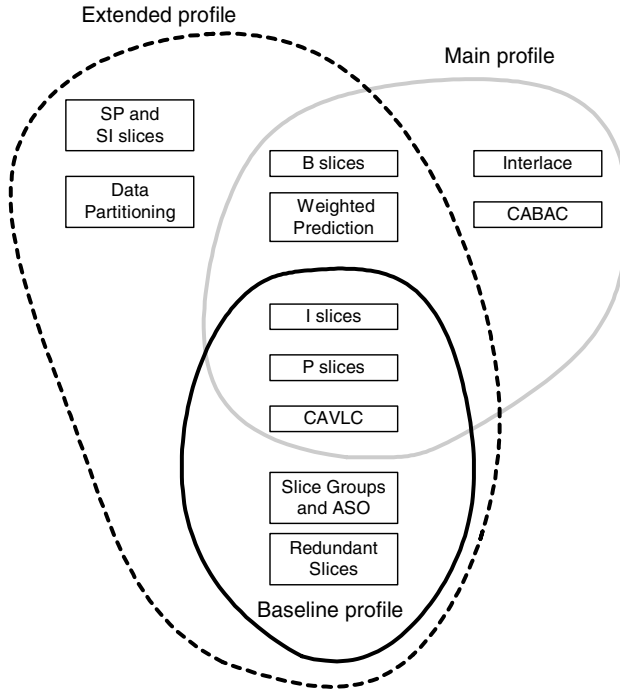


Figure 6.3 H.264 Baseline, Main and Extended profiles

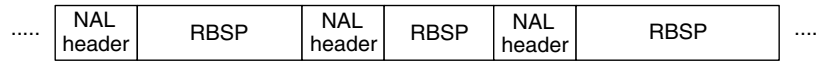


Figure 6.4 Sequence of NAL units

6.3.3 Coded Data Format

H.264 makes a distinction between a Video Coding Layer (VCL) and a Network Abstraction Layer (NAL). The output of the encoding process is VCL data (a sequence of bits representing the coded video data) which are mapped to NAL units prior to transmission or storage. Each NAL unit contains a Raw Byte Sequence Payload (RBSP), a set of data corresponding to coded video data or header information. A coded video sequence is represented by a sequence of NAL units (Figure 6.4) that can be transmitted over a packet-based network or a bitstream transmission link or stored in a file. The purpose of separately specifying the VCL and NAL is to distinguish between coding-specific features (at the VCL) and transport-specific features (at the NAL). Section 6.7 describes the NAL and transport mechanisms in more detail.

6.3.4 Reference Pictures

An H.264 encoder may use one or two of a number of previously encoded pictures as a reference for motion-compensated prediction of each inter coded macroblock or macroblock

Table 6.1 H.264 slice modes

Slice type	Description	Profile(s)
I (Intra)	Contains only I macroblocks (each block or macroblock is predicted from previously coded data within the same slice).	All
P (Predicted)	Contains P macroblocks (each macroblock or macroblock partition is predicted from one list 0 reference picture) and/or I macroblocks.	All
B (Bi-predictive)	Contains B macroblocks (each macroblock or macroblock partition is predicted from a list 0 and/or a list 1 reference picture) and/or I macroblocks.	Extended and Main
SP (Switching P)	Facilitates switching between coded streams; contains P and/or I macroblocks.	Extended
SI (Switching I)	Facilitates switching between coded streams; contains SI macroblocks (a special type of intra coded macroblock).	Extended

partition. This enables the encoder to search for the best 'match' for the current macroblock partition from a wider set of pictures than just (say) the previously encoded picture.

The encoder and decoder each maintain one or two lists of reference pictures, containing pictures that have previously been encoded and decoded (occurring before and/or after the current picture in display order). Inter coded macroblocks and macroblock partitions in P slices (see below) are predicted from pictures in a single list, **list 0**. Inter coded macroblocks and macroblock partitions in a B slice (see below) may be predicted from two lists, **list 0** and **list 1**.

6.3.5 Slices

A video picture is coded as one or more slices, each containing an integral number of macroblocks from 1 (1 MB per slice) to the total number of macroblocks in a picture (1 slice per picture) The number of macroblocks per slice need not be constant within a picture. There is minimal inter-dependency between coded slices which can help to limit the propagation of errors. There are five types of coded slice (Table 6.1) and a coded picture may be composed of different types of slices. For example, a Baseline Profile coded picture may contain a mixture of I and P slices and a Main or Extended Profile picture may contain a mixture of I, P and B slices.

Figure 6.5 shows a simplified illustration of the syntax of a coded slice. The slice header defines (among other things) the slice type and the coded picture that the slice 'belongs' to and may contain instructions related to reference picture management (see Section 6.4.2). The slice data consists of a series of coded macroblocks and/or an indication of skipped (not coded) macroblocks. Each MB contains a series of header elements (see Table 6.2) and coded residual data.

6.3.6 Macroblocks

A macroblock contains coded data corresponding to a 16 × 16 sample region of the video frame (16 × 16 luma samples, 8 × 8 Cb and 8 × 8 Cr samples) and contains the syntax elements described in Table 6.2. Macroblocks are numbered (addressed) in raster scan order within a frame.

Table 6.2 Macroblock syntax elements

mb_type	Determines whether the macroblock is coded in intra or inter (P or B) mode; determines macroblock partition size (see Section 6.4.2).
mb_pred	Determines intra prediction modes (intra macroblocks); determines list 0 and/or list 1 references and differentially coded motion. vectors for each macroblock partition (inter macroblocks, except for inter MBs with 8 × 8 macroblock partition size).
sub_mb_pred	(Inter MBs with 8 × 8 macroblock partition size only) Determines sub-macroblock partition size for each sub-macroblock; list 0 and/or list 1 references for each macroblock partition; differentially coded motion vectors for each macroblock sub-partition.
coded_block_pattern	Identifies which 8 × 8 blocks (luma and chroma) contain coded transform coefficients.
mb_qp_delta	Changes the quantiser parameter (see Section 6.4.8).
residual	Coded transform coefficients corresponding to the residual image samples after prediction (see Section 6.4.8).

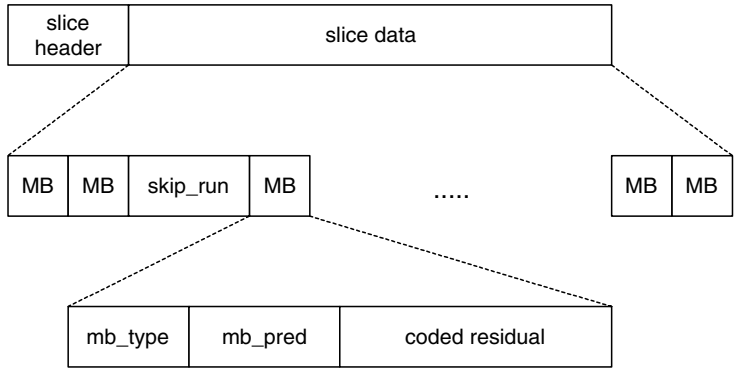


Figure 6.5 Slice syntax

6.4 THE BASELINE PROFILE

6.4.1 Overview

The Baseline Profile supports coded sequences containing I- and P-slices. I-slices contain intra-coded macroblocks in which each 16 × 16 or 4 × 4 luma region and each 8 × 8 chroma region is predicted from previously-coded samples in the same slice. P-slices may contain intra-coded, inter-coded or skipped MBs. Inter-coded MBs in a P slice are predicted from a number of previously coded pictures, using motion compensation with quarter-sample (luma) motion vector accuracy.

After prediction, the residual data for each MB is transformed using a 4 × 4 integer transform (based on the DCT) and quantised. Quantised transform coefficients are reordered and the syntax elements are entropy coded. In the Baseline Profile, transform coefficients are entropy coded using a context-adaptive variable length coding scheme (CAVLC) and all other

syntax elements are coded using fixed-length or Exponential-Golomb Variable Length Codes. Quantised coefficients are scaled, inverse transformed, reconstructed (added to the prediction formed during encoding) and filtered with a de-blocking filter before (optionally) being stored for possible use in reference pictures for further intra- and inter-coded macroblocks.

6.4.2 Reference Picture Management

Pictures that have previously been encoded are stored in a reference buffer (the decoded picture buffer, DPB) in both the encoder and the decoder. The encoder and decoder maintain a list of previously coded pictures, reference picture list 0, for use in motion-compensated prediction of inter macroblocks in P slices. For P slice prediction, list 0 can contain pictures before and after the current picture in display order and may contain both *short term* and *long term* reference pictures. By default, an encoded picture is reconstructed by the encoder and marked as a short term picture, a recently-coded picture that is available for prediction. Short term pictures are identified by their frame number. Long term pictures are (typically) older pictures that may also be used for prediction and are identified by a variable LongTermPicNum. Long term pictures remain in the DPB until explicitly removed or replaced.

When a picture is encoded and reconstructed (in the encoder) or decoded (in the decoder), it is placed in the decoded picture buffer and is either (a) marked as 'unused for reference' (and hence not used for any further prediction), (b) marked as a short term picture, (c) marked as a long term picture or (d) simply output to the display. By default, short term pictures in list 0 are ordered from the highest to the lowest PicNum (a variable derived from the frame number) and long term pictures are ordered from the lowest to the highest LongTermPicNum. The encoder may signal a change to the default reference picture list order. As each new picture is added to the short term list at position 0, the indices of the remaining short-term pictures are incremented. If the number of short term and long term pictures is equal to the maximum number of reference frames, the oldest short-term picture (with the highest index) is removed from the buffer (known as *sliding window* memory control). The effect that this process is that the encoder and decoder each maintain a 'window' of N short-term reference pictures, including the current picture and $(N - 1)$ previously encoded pictures.

Adaptive memory control commands, sent by the encoder, manage the short and long term picture indexes. Using these commands, a short term picture may be assigned a long term frame index, or any short term or long term picture may be marked as 'unused for reference'.

The encoder chooses a reference picture from list 0 for encoding each macroblock partition in an inter-coded macroblock. The choice of reference picture is signalled by an index number, where index 0 corresponds to the first frame in the short term section and the indices of the long term frames start after the last short term frame (as shown in the following example).

Example: Reference buffer management (P-slice)

Current frame number = 250

Number of reference frames = 5

	Reference picture list				
Operation	0	1	2	3	4
Initial state	–	–	–	–	–
Encode frame 250	250	–	–	–	–
Encode 251	251	250	–	–	–
Encode 252	252	251	250	–	–
Encode 253	253	252	251	250	–
Assign 251 to LongTermPicNum 0	253	252	250	0	–
Encode 254	254	253	252	250	0
Assign 253 to LongTermPicNum 4	254	252	250	0	4
Encode 255	255	254	252	0	4
Assign 255 to LongTermPicNum 3	254	252	0	3	4
Encode 256	256	254	0	3	4

(Note that in the above example, 0, 3 and 4 correspond to the decoded frames 251, 255 and 253 respectively).

Instantaneous Decoder Refresh Picture

An encoder sends an IDR (Instantaneous Decoder Refresh) coded picture (made up of I- or SI-slices) to clear the contents of the reference picture buffer. On receiving an IDR coded picture, the decoder marks all pictures in the reference buffer as 'unused for reference'. All subsequent transmitted slices can be decoded without reference to any frame decoded prior to the IDR picture. The first picture in a coded video sequence is always an IDR picture.

6.4.3 Slices

A bitstream conforming to the the Baseline Profile contains coded I and/or P slices. An I slice contains only intra-coded macroblocks (predicted from previously coded samples in the same slice, see Section 6.4.6) and a P slice can contain inter coded macroblocks (predicted from samples in previously coded pictures, see Section 6.4.5), intra coded macroblocks or Skipped macroblocks. When a Skipped macroblock is signalled in the bitstream, no further data is sent for that macroblock. The decoder calculates a vector for the skipped macroblock (see Section 6.4.5.3) and reconstructs the macroblock using motion-compensated prediction from the first reference picture in list 0.

An H.264 encoder may optionally insert a picture delimiter RBSP unit at the boundary between coded pictures. This indicates the start of a new coded picture and indicates which slice types are allowed in the following coded picture. If the picture delimiter is not used, the decoder is expected to detect the occurrence of a new picture based on the header of the first slice in the new picture.

Redundant coded picture

A picture marked as 'redundant' contains a redundant representation of part or all of a coded picture. In normal operation, the decoder reconstructs the frame from 'primary'

Table 6.3 Macroblock to slice group map types

Type	Name	Description
0	Interleaved	run_length MBs are assigned to each slice group in turn (Figure 6.6).
1	Dispersed	MBs in each slice group are dispersed throughout the picture (Figure 6.7).
2	Foreground and background	All but the last slice group are defined as rectangular regions within the picture. The last slice group contains all MBs not contained in any other slice group (the 'background'). In the example in Figure 6.8, group 1 overlaps group 0 and so MBs not already allocated to group 0 are allocated to group 1.
3	Box-out	A 'box' is created starting from the centre of the frame (with the size controlled by encoder parameters) and containing group 0; all other MBs are in group 1 (Figure 6.9).
4	Raster scan	Group 0 contains MBs in raster scan order from the top-left and all other MBs are in group 1 (Figure 6.9).
5	Wipe	Group 0 contains MBs in vertical scan order from the top-left and all other MBs are in group 1 (Figure 6.9).
6	Explicit	A parameter, slice_group_id, is sent for each MB to indicate its slice group (i.e. the macroblock map is entirely user-defined).

(nonredundant)' pictures and discards any redundant pictures. However, if a primary coded picture is damaged (e.g. due to a transmission error), the decoder may replace the damaged area with decoded data from a redundant picture if available.

Arbitrary Slice Order (ASO)

The Baseline Profile supports Arbitrary Slice Order which means that slices in a coded frame may follow any decoding order. ASO is defined to be in use if the first macroblock in any slice in a decoded frame has a smaller macroblock address than the first macroblock in a *previously* decoded slice in the same picture.

Slice Groups

A *slice group* is a subset of the macroblocks in a coded picture and may contain one or more slices. Within each slice in a slice group, MBs are coded in raster order. If only one slice group is used per picture, then all macroblocks in the picture are coded in raster order (unless ASO is in use, see above). Multiple slice groups (described in previous versions of the draft standard as Flexible Macroblock Ordering or FMO) make it possible to map the sequence of coded MBs to the decoded picture in a number of flexible ways. The allocation of macroblocks is determined by a *macroblock to slice group map* that indicates which slice group each MB belongs to. Table 6.3 lists the different types of macroblock to slice group maps.

Example: 3 slice groups are used and the map type is 'interleaved' (Figure 6.6). The coded picture consists of first, all of the macroblocks in slice group 0 (filling every 3rd row of macroblocks); second, all of the macroblocks in slice group 1; and third, all of the macroblocks in slice group 0. Applications of multiple slice groups include error resilience, for example if one of the slice groups in the dispersed map shown in Figure 6.7 is 'lost' due to errors, the missing data may be concealed by interpolation from the remaining slice groups.

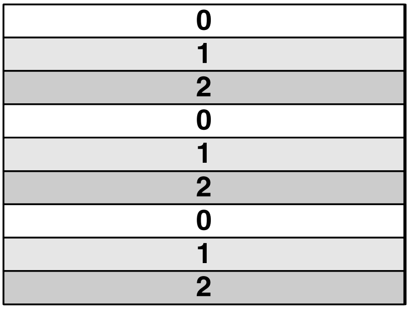


Figure 6.6 Slice groups: Interleaved map (QCIF, three slice groups)

0	1	2	3	0	1	2	3	0	1	2
2	3	0	1	2	3	0	1	2	3	0
0	1	2	3	0	1	2	3	0	1	2
2	3	0	1	2	3	0	1	2	3	0
0	1	2	3	0	1	2	3	0	1	2
2	3	0	1	2	3	0	1	2	3	0
0	1	2	3	0	1	2	3	0	1	2
2	3	0	1	2	3	0	1	2	3	0
0	1	2	3	0	1	2	3	0	1	2

Figure 6.7 Slice groups: Dispersed map (QCIF, four slice groups)

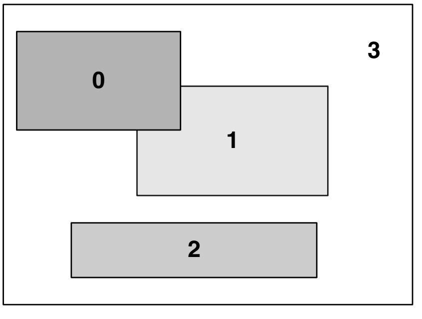


Figure 6.8 Slice groups: Foreground and Background map (four slice groups)

6.4.4 Macroblock Prediction

Every coded macroblock in an H.264 slice is predicted from previously-encoded data. Samples within an intra macroblock are predicted from samples in the current slice that have already been encoded, decoded and reconstructed; samples in an inter macroblock are predicted from previously-encoded.

A prediction for the current macroblock or block (a model that resembles the current macroblock or block as closely as possible) is created from image samples that have already

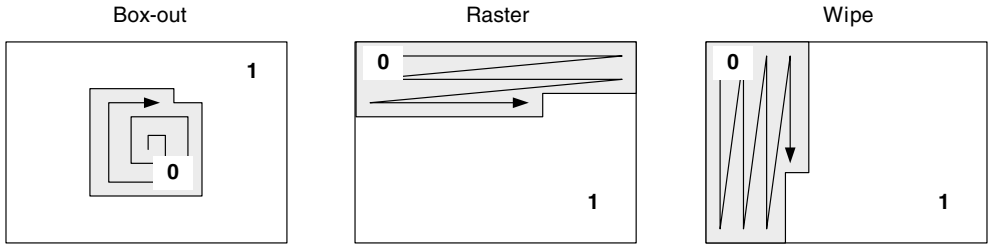


Figure 6.9 Slice groups: Box-out, Raster and Wipe maps

been encoded (either in the same slice or in a previously encoded slice). This prediction is subtracted from the current macroblock or block and the result of the subtraction (residual) is compressed and transmitted to the decoder, together with information required for the decoder to repeat the prediction process (motion vector(s), prediction mode, etc.). The decoder creates an identical prediction and adds this to the decoded residual or block. The encoder bases its prediction on encoded and decoded image samples (rather than on original video frame samples) in order to ensure that the encoder and decoder predictions are identical.

6.4.5 Inter Prediction

Inter prediction creates a prediction model from one or more previously encoded video frames or fields using block-based motion compensation. Important differences from earlier standards include the support for a range of block sizes (from 16×16 down to 4×4) and fine sub-sample motion vectors (quarter-sample resolution in the luma component). In this section we describe the inter prediction tools available in the Baseline profile. Extensions to these tools in the Main and Extended profiles include B-slices (Section 6.5.1) and Weighted Prediction (Section 6.5.2).

6.4.5.1 Tree structured motion compensation

The luminance component of each macroblock (16×16 samples) may be split up in four ways (Figure 6.10) and motion compensated either as one 16×16 *macroblock partition*, two 16×8 partitions, two 8×16 partitions or four 8×8 partitions. If the 8×8 mode is chosen, each of the four 8×8 sub-macroblocks within the macroblock may be split in a further 4 ways (Figure 6.11), either as one 8×8 sub-macroblock partition, two 8×4 sub-macroblock partitions, two 4×8 sub-macroblock partitions or four 4×4 sub-macroblock partitions. These partitions and sub-macroblock give rise to a large number of possible combinations within each macroblock. This method of partitioning macroblocks into motion compensated sub-blocks of varying size is known as *tree structured motion compensation*.

A separate motion vector is required for each partition or sub-macroblock. Each motion vector must be coded and transmitted and the choice of partition(s) must be encoded in the compressed bitstream. Choosing a large partition size (16×16, 16×8, 8×16) means that

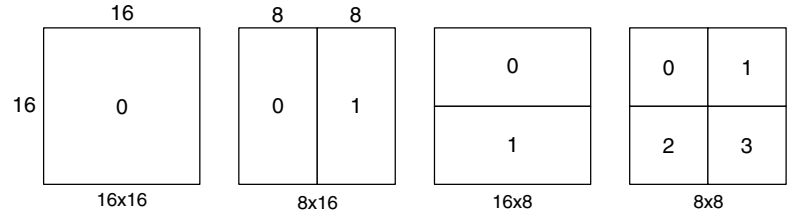


Figure 6.10 Macroblock partitions: 16 × 16, 8 × 16, 16 × 8, 8 × 8

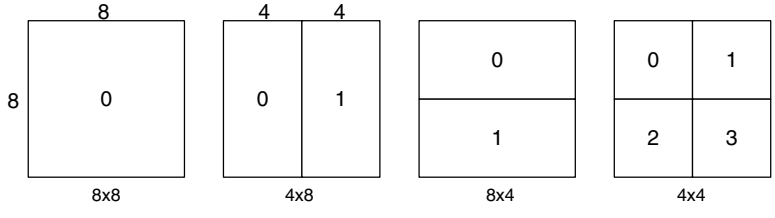


Figure 6.11 Sub-macroblock partitions: 8 × 8, 4 × 8, 8 × 4, 4 × 4

a small number of bits are required to signal the choice of motion vector(s) and the type of partition but the motion compensated residual may contain a significant amount of energy in frame areas with high detail. Choosing a small partition size (8 × 4, 4 × 4, etc.) may give a lower-energy residual after motion compensation but requires a larger number of bits to signal the motion vectors and choice of partition(s). The choice of partition size therefore has a significant impact on compression performance. In general, a large partition size is appropriate for homogeneous areas of the frame and a small partition size may be beneficial for detailed areas.

Each chroma component in a macroblock (Cb and Cr) has half the horizontal and vertical resolution of the luminance (luma) component. Each chroma block is partitioned in the same way as the luma component, except that the partition sizes have exactly half the horizontal and vertical resolution (an 8 × 16 partition in luma corresponds to a 4 × 8 partition in chroma; an 8 × 4 partition in luma corresponds to 4 × 2 in chroma and so on). The horizontal and vertical components of each motion vector (one per partition) are halved when applied to the chroma blocks.

Example

Figure 6.12 shows a residual frame (without motion compensation). The H.264 reference encoder selects the 'best' partition size for each part of the frame, in this case the partition size that minimises the amount of information to be sent, and the chosen partitions are shown superimposed on the residual frame. In areas where there is little change between the frames (residual appears grey), a 16 × 16 partition is chosen and in areas of detailed motion (residual appears black or white), smaller partitions are more efficient.

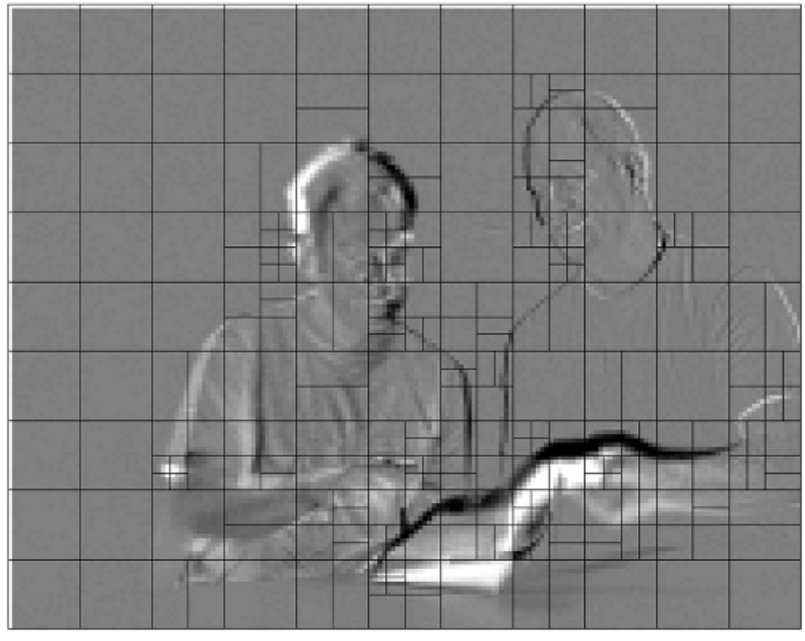


Figure 6.12 Residual (without MC) showing choice of block sizes

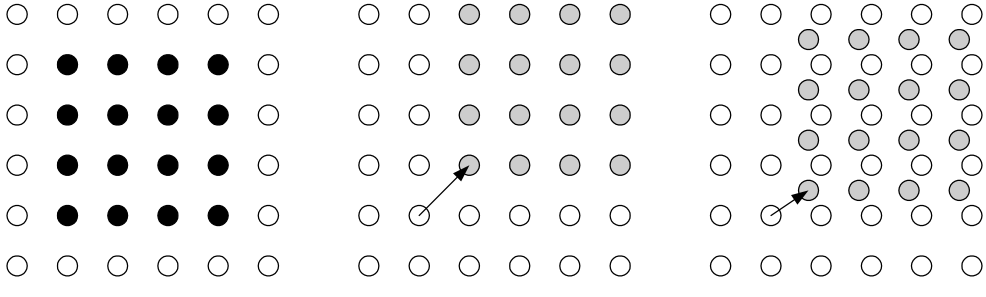


Figure 6.13 Example of integer and sub-sample prediction

6.4.5.2 Motion Vectors

Each partition or sub-macroblock partition in an inter-coded macroblock is predicted from an area of the same size in a reference picture. The offset between the two areas (the motion vector) has quarter-sample resolution for the luma component and one-eighth-sample resolution for the chroma components. The luma and chroma samples at sub-sample positions do not exist in the reference picture and so it is necessary to create them using interpolation from nearby coded samples. In Figure 6.13, a 4×4 block in the current frame (a) is predicted from a region of the reference picture in the neighbourhood of the current block position. If the horizontal and vertical components of the motion vector are integers (b), the relevant samples in the

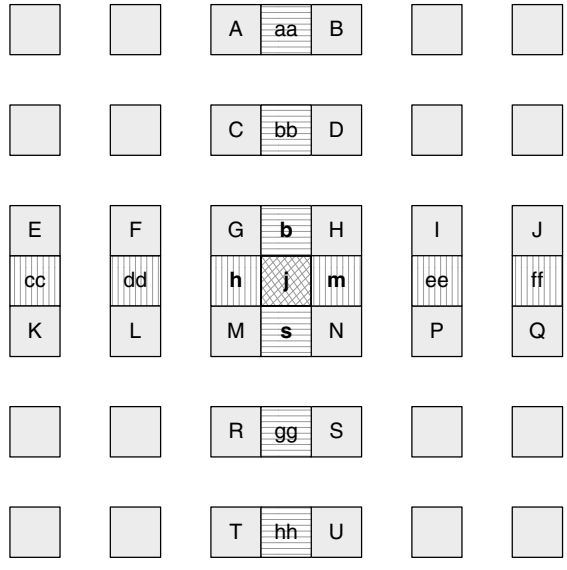


Figure 6.14 Interpolation of luma half-pel positions

reference block actually exist (grey dots). If one or both vector components are fractional values (c), the prediction samples (grey dots) are generated by interpolation between adjacent samples in the reference frame (white dots).

Generating Interpolated Samples

The samples half-way between integer-position samples ('half-pel samples') in the luma component of the reference picture are generated first (Figure 6.14, grey markers). Each half-pel sample that is adjacent to two integer samples (e.g. b, h, m, s in Figure 6.14) is interpolated from integer-position samples using a six tap Finite Impulse Response (FIR) filter with weights $(1/32, -5/32, 5/8, 5/8, -5/32, 1/32)$. For example, half-pel sample **b** is calculated from the six horizontal integer samples E, F, G, H, I and J:

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

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

Once all the half-pel samples are available, the samples at quarter-step ('quarter-pel') positions are produced by linear interpolation (Figure 6.15). Quarter-pel positions with two horizontally or vertically adjacent half- or integer-position samples (e.g. **a**, **c**, **i**, **k** and **d**, **f**, **n**,

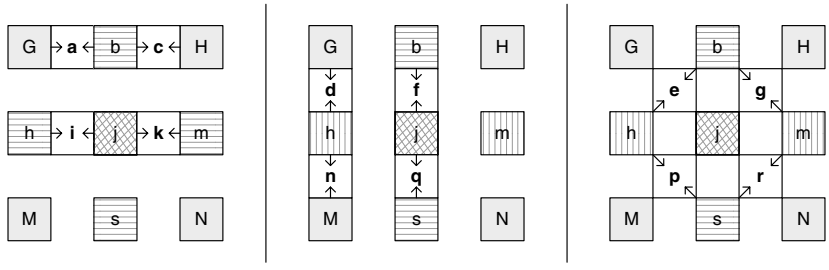


Figure 6.15 Interpolation of luma quarter-pel positions

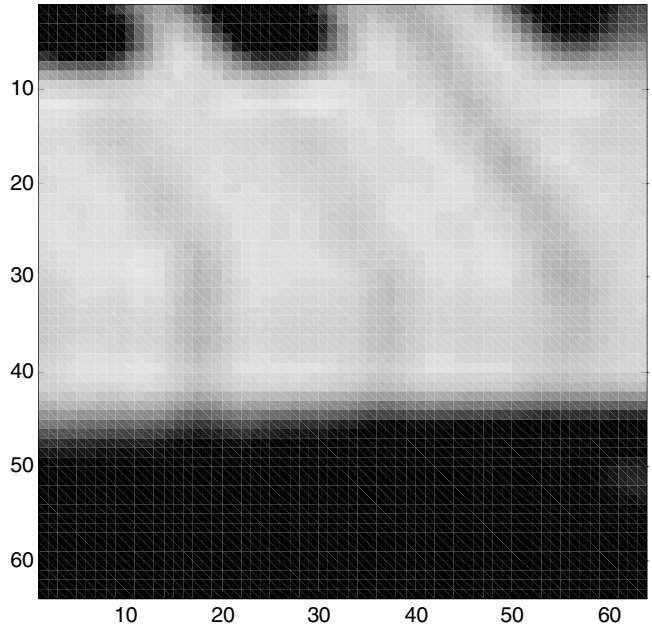


Figure 6.16 Luma region interpolated to quarter-pel positions

q in Figure 6.15) are linearly interpolated between these adjacent samples, for example:

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

The remaining quarter-pel positions (**e**, **g**, **p** and **r** in the figure) are linearly interpolated between a pair of diagonally opposite *half*-pel samples. For example, **e** is interpolated between b and h. Figure 6.16 shows the result of interpolating the reference region shown in Figure 3.16 with quarter-pel resolution.

Quarter-pel resolution motion vectors in the luma component require eighth-sample resolution vectors in the chroma components (assuming 4:2:0 sampling). Interpolated samples are generated at eighth-sample intervals between integer samples in each chroma component using linear interpolation (Figure 6.17). Each sub-sample position **a** is a linear combination

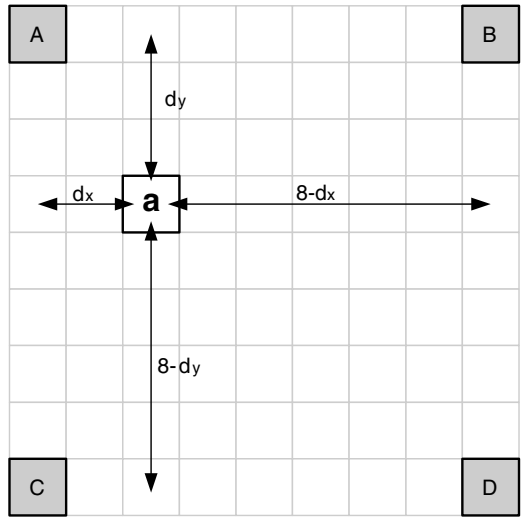


Figure 6.17 Interpolation of chroma eighth-sample positions

of the neighbouring integer sample positions A, B, C and D:

$$\mathbf{a} = \text{round}([(8 - d_x) \cdot (8 - d_y)\text{A} + d_x \cdot (8 - d_y)\text{B} + (8 - d_x) \cdot d_y\text{C} + d_x \cdot d_y\text{D}]/64)$$

In Figure 6.17, d_x is 2 and d_y is 3, so that:

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

6.4.5.3 Motion Vector Prediction

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

Let E be the current macroblock, macroblock partition or sub-macroblock partition, let A be the partition or sub-partition immediately to the left of E, let B be the partition or sub-partition immediately above E and let C be the partition or sub-macroblock partition above and to the right of E. If there is more than one partition immediately to the left of E, the topmost of these partitions is chosen as A. If there is more than one partition immediately above E, the leftmost of these is chosen as B. Figure 6.18 illustrates the choice of neighbouring partitions when all the partitions have the same size (16×16 in this case) and Figure 6.19 shows an

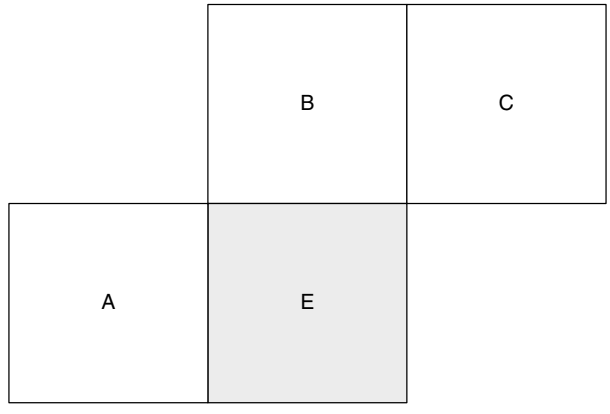


Figure 6.18 Current and neighbouring partitions (same partition sizes)

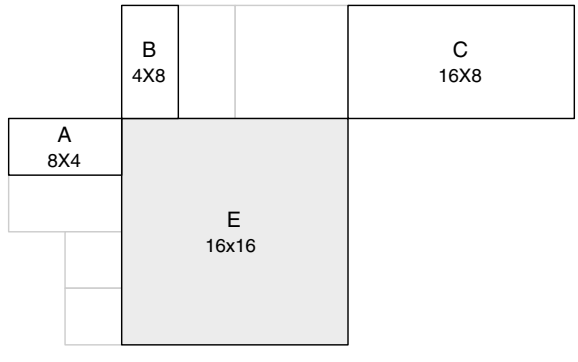


Figure 6.19 Current and neighbouring partitions (different partition sizes)

example of the choice of prediction partitions when the neighbouring partitions have different sizes from the current partition E.

1. For transmitted partitions excluding 16 × 8 and 8 × 16 partition sizes, MVp is the median of the motion vectors for partitions A, B and C.
2. For 16 × 8 partitions, MVp for the upper 16 × 8 partition is predicted from B and MVp for the lower 16 × 8 partition is predicted from A.
3. For 8 × 16 partitions, MVp for the left 8 × 16 partition is predicted from A and MVp for the right 8 × 16 partition is predicted from C.
4. For skipped macroblocks, a 16 × 16 vector MVp is generated as in case (1) above (i.e. as if the block were encoded in 16 × 16 Inter mode).

If one or more of the previously transmitted blocks shown in Figure 6.19 is not available (e.g. if it is outside the current slice), the choice of MVp is modified accordingly. At the decoder, the predicted vector MVp is formed in the same way and added to the decoded vector difference MVD. In the case of a skipped macroblock, there is no decoded vector difference and a motion-compensated macroblock is produced using MVp as the motion vector.

Figure 6.20 QCIF frame

6.4.6 Intra Prediction

In intra mode a prediction block P is formed based on previously encoded and reconstructed blocks and is subtracted from the current block prior to encoding. For the luma samples, P is formed for each 4 × 4 block or for a 16 × 16 macroblock. There are a total of nine optional prediction modes for each 4 × 4 luma block, four modes for a 16 × 16 luma block and four modes for the chroma components. The encoder typically selects the prediction mode for each block that minimises the difference between P and the block to be encoded.

Example

A QCIF video frame (Figure 6.20) is encoded in intra mode and each block or macroblock is predicted from neighbouring, previously-encoded samples. Figure 6.21 shows the predicted luma frame P formed by choosing the 'best' 4 × 4 or 16 × 16 prediction mode for each region (the mode that minimises the amount of information to be coded).

A further intra coding mode, I_PCM, enables an encoder to transmit the values of the image samples directly (without prediction or transformation). In some special cases (e.g. anomalous image content and/or very low quantizer parameters), this mode may be more efficient than the 'usual' process of intra prediction, transformation, quantization and entropy coding. Including the I_PCM option makes it possible to place an absolute limit on the number of bits that may be contained in a coded macroblock without constraining decoded image quality.

6.4.6.1 4 × 4 Luma Prediction Modes

Figure 6.22 shows a 4 × 4 luma block (part of the highlighted macroblock in Figure 6.20) that is required to be predicted. The samples above and to the left (labelled A–M in Figure 6.23)

Figure 6.21 Predicted luma frame formed using H.264 intra prediction

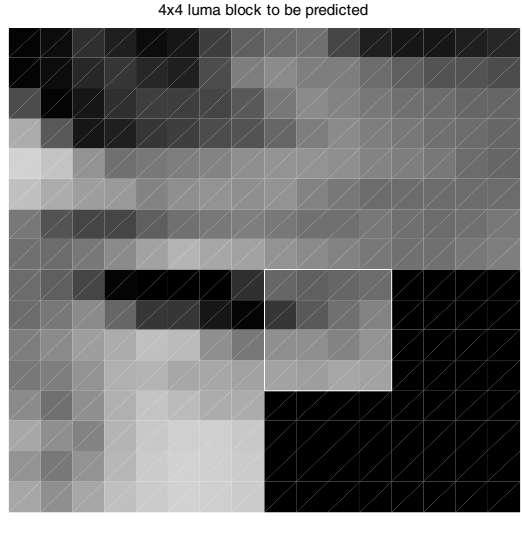


Figure 6.22 4 × 4 luma block to be predicted

have previously been encoded and reconstructed and are therefore available in the encoder and decoder to form a prediction reference. The samples a, b, c, . . . , p of the prediction block P (Figure 6.23) are calculated based on the samples A–M as follows. Mode 2 (DC prediction) is modified depending on which samples A–M have previously been coded; each of the other modes may only be used if all of the required prediction samples are available[2].

[2] Note that if samples E, F, G and H have not yet been decoded, the value of sample D is copied to these positions and they are marked as 'available'.

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

Figure 6.23 Labelling of prediction samples (4 × 4)

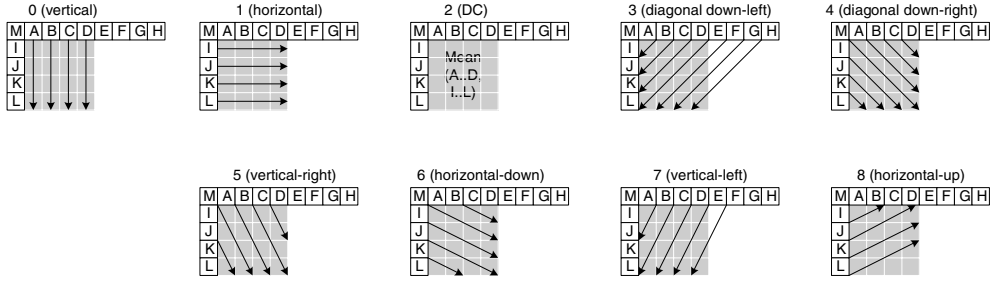


Figure 6.24 4 × 4 luma prediction modes

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

The arrows in Figure 6.24 indicate the direction of prediction in each mode. For modes 3–8, the predicted samples are formed from a *weighted average* of the prediction samples A–M. For example, if mode 4 is selected, the top-right sample of P (labelled 'd' in Figure 6.23) is predicted by: round(B/4+ C/2+ D/4).

Example

The nine prediction modes (0–8) are calculated for the 4 × 4 block shown in Figure 6.22 and the resulting prediction blocks P are shown in Figure 6.25. The Sum of Absolute Errors (SAE) for each prediction indicates the magnitude of the prediction error. In this case, the best match to

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

6.4.6.2 16 × 16 Luma Prediction Modes

As an alternative to the 4 × 4 luma modes described in the previous section, the entire 16 × 16 luma component of a macroblock may be predicted in one operation. Four modes are available, shown in diagram form in Figure 6.26:

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

Example:

Figure 6.27 shows a luminance macroblock with previously-encoded samples at the upper and left-hand edges. The results of the four prediction modes, shown in Figure 6.28, indicate that the best match is given by mode 3 which in this case produces a plane with a luminance gradient from light (upper-left) to dark (lower-right). Intra 16 × 16 mode works best in homogeneous areas of an image.

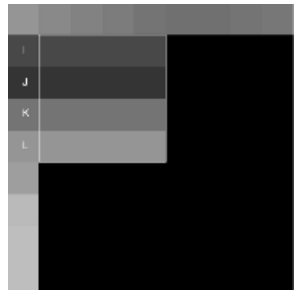
6.4.6.3 8 × 8 Chroma Prediction Modes

Each 8 × 8 chroma component of an intra coded a macroblock is predicted from previously encoded chroma samples above and/or to the left and both chroma components always use the same prediction mode. The four prediction modes are very similar to the 16 × 16 luma prediction modes described in Section 6.4.6.2 and illustrated in Figure 6.26, except that the numbering of the modes is different. The modes are DC (mode 0), horizontal (mode 1), vertical (mode 2) and plane (mode 3).

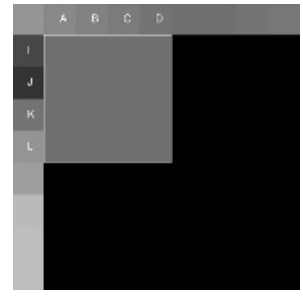
6.4.6.4 Signalling Intra Prediction Modes

The choice of intra prediction mode for each 4 × 4 block must be signalled to the decoder and this could potentially require a large number of bits. However, intra modes for neighbouring 4 × 4 blocks are often correlated. For example, let A, B and E be the left, upper and current 4 × 4 blocks respectively (the same labelling as Figure 6.18). If previously-encoded 4 × 4 blocks A and B are predicted using mode 1, it is probable that the best mode for block E (current block) is also mode 1. To take advantage of this correlation, predictive coding is used to signal 4 × 4 intra modes.

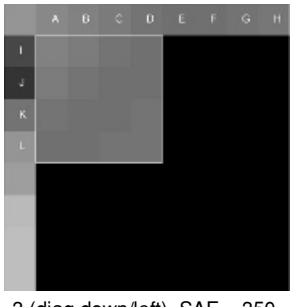
0 (vertical), SAE = 317



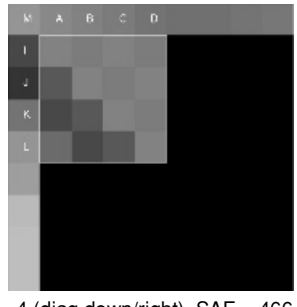
1 (horizontal), SAE = 401



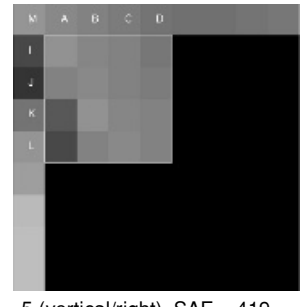
2 (DC), SAE = 317



3 (diag down/left), SAE = 350

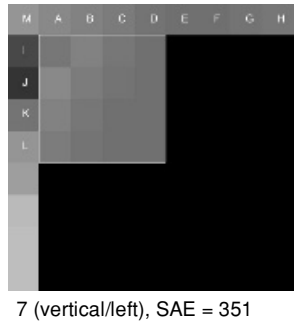


4 (diag down/right), SAE = 466

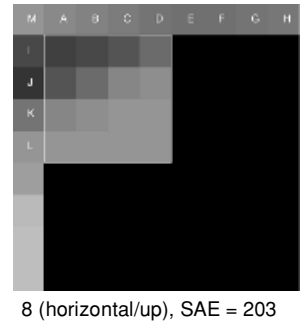


5 (vertical/right), SAE = 419

6 (horizontal/down), SAE = 530



7 (vertical/left), SAE = 351



8 (horizontal/up), SAE = 203

Figure 6.25 Prediction blocks, luma 4 × 4

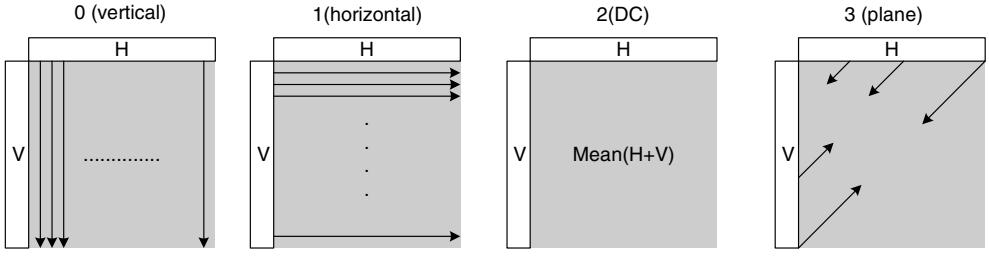


Figure 6.26 Intra 16 × 16 prediction modes

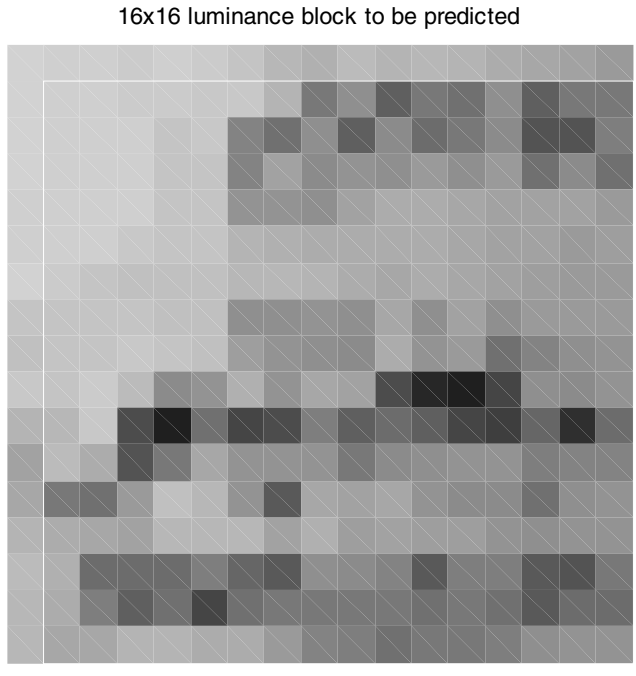


Figure 6.27 16 × 16 macroblock

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

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

Example

Blocks A and B have been predicted using mode 3 (diagonal down-left) and 1 (horizontal) respectively. The most probable mode for block E is therefore 1 (horizontal). prev_intra4×4_pred_mode is set to '0' and so rem_intra4×4_pred_mode is sent. Depending on the value of rem_intra4×4_pred_mode, one of the eight remaining prediction modes (listed in Table 6.4) may be chosen.

The prediction mode for luma coded in Intra-16×16 mode or chroma coded in Intra mode is signalled in the macroblock header and predictive coding of the mode is not used in these cases.

Table 6.4 Choice of prediction mode (most probable mode = 1)

rem_intra4×4_pred_mode	prediction mode for block C
0	0
1	2
2	3
3	4
4	5
5	6
6	7
7	8

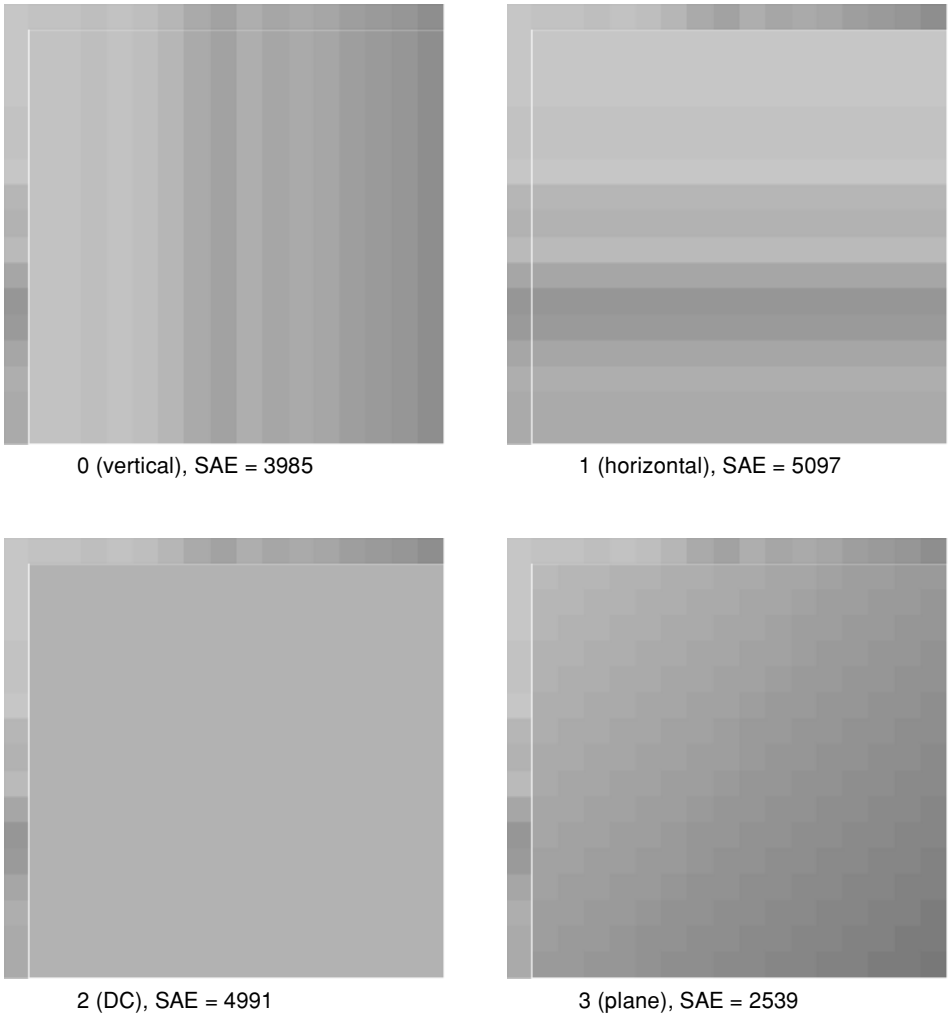


Figure 6.28 Prediction blocks, intra 16 × 16

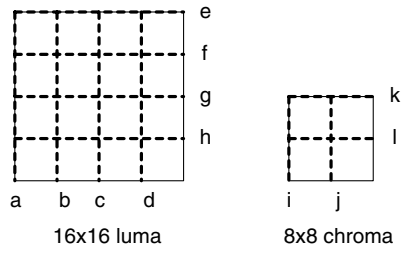


Figure 6.29 Edge filtering order in a macroblock

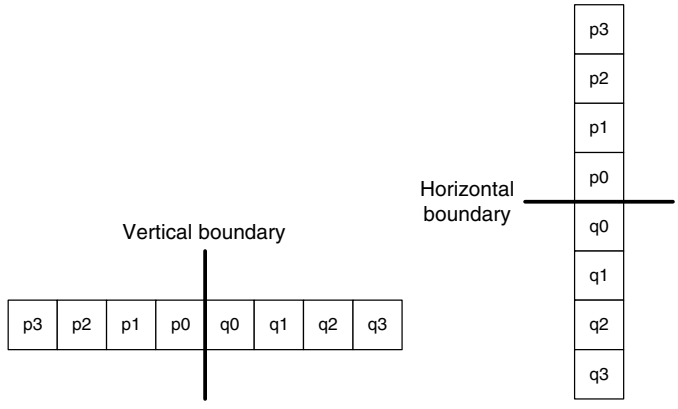


Figure 6.30 Samples adjacent to vertical and horizontal boundaries

6.4.7 Deblocking Filter

A filter is applied to each decoded macroblock to reduce blocking distortion. The deblocking filter is applied after the inverse transform in the encoder (before reconstructing and storing the macroblock for future predictions) and in the decoder (before reconstructing and displaying the macroblock). The filter smooths block edges, improving the appearance of decoded frames. The filtered image is used for motion-compensated prediction of future frames and this can improve compression performance because the filtered image is often a more faithful reproduction of the original frame than a blocky, unfiltered image[3]. The default operation of the filter is as follows; it is possible for the encoder to alter the filter strength or to disable the filter.

Filtering is applied to vertical or horizontal edges of 4×4 blocks in a macroblock (except for edges on slice boundaries), in the following order.

1. Filter 4 vertical boundaries of the luma component (in order a, b, c, d in Figure 6.29).
2. Filter 4 horizontal boundaries of the luma component (in order e, f, g, h, Figure 6.29).
3. Filter 2 vertical boundaries of each chroma component (i, j).
4. Filter 2 horizontal boundaries of each chroma component (k, l).

Each filtering operation affects up to *three* samples on either side of the boundary. Figure 6.30 shows four samples on either side of a vertical or horizontal boundary in adjacent blocks

[3] Intra-coded macroblocks are filtered, but intra prediction (Section 6.4.6) is carried out using *unfiltered* reconstructed macroblocks to form the prediction.

p and q (p0, p1, p2, p3 and q0, q1, q2, q3). The 'strength' of the filter (the amount of filtering) depends on the current quantiser, the coding modes of neighbouring blocks and the gradient of image samples across the boundary.

Boundary Strength

The choice of filtering outcome depends on the *boundary strength* and on the *gradient* of image samples across the boundary. The boundary strength parameter *bS* is chosen according to the following rules (for coding of progressive frames):

p and/or q is intra coded *and* boundary is a macroblock boundary	$bS = 4$ (strongest filtering)
p and q are intra coded and boundary is *not* a macroblock boundary	$bS = 3$
neither p or q is intra coded; p and q contain coded coefficients	$bS = 2$
neither p or q is intra coded; neither p or q contain coded coefficients; p and q use different reference pictures *or* a different number of reference pictures or have motion vector values that differ by one luma sample or more	$bS = 1$
otherwise	$bS = 0$ (no filtering)

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

Filter Decision

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

(a) $BS > 0$ and
(b) $|p0-q0| < \alpha$ *and* $|p1-p0| < \beta$ *and* $|q1-q0| \leq \beta$.

α and β are thresholds defined in the standard; they increase with the average quantiser parameter QP of the two blocks p and q. The effect of the filter decision is to 'switch off' the filter when there is a significant change (gradient) across the block boundary in the original image. When QP is small, anything other than a very small gradient across the boundary is likely to be due to image features (rather than blocking effects) that should be preserved and so the thresholds α and β are low. When QP is larger, blocking distortion is likely to be more significant and α, β are higher so that more boundary samples are filtered.

Example

Figure 6.31 shows the 16 × 16 luma component of a macroblock (without any blocking distortion) with four 4 × 4 blocks a, b, c and d highlighted. Assuming a medium to large value of QP, the block boundary between a and b is likely to be filtered because the gradient across this boundary is small. There are no significant image features to preserve and blocking distortion will be obvious on this boundary. However, there is a significant change in luminance across the boundary between c and d due to a horizontal image feature and so the filter is switched off to preserve this strong feature.

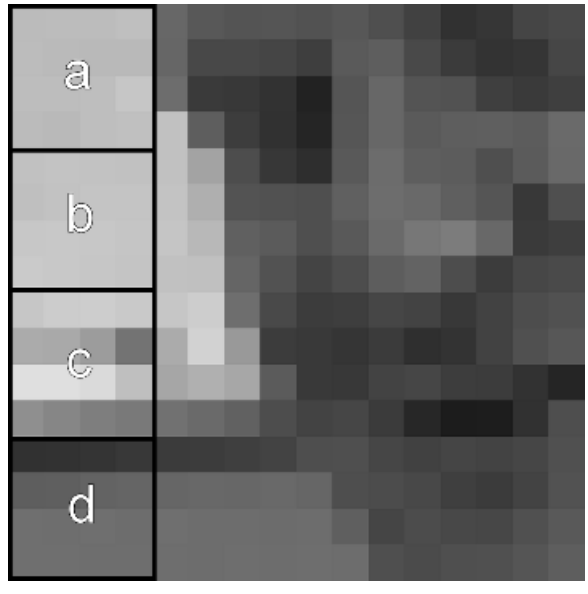


Figure 6.31 16 × 16 luma macroblock showing block edges

Filter Implementation

(a) bS ∈ {1,2,3}

A 4-tap filter is applied with inputs p1, p0, q0 and q1, producing filtered outputs p'0 and q'0. If |p2 − p0| is less than threshold β, another four-tap filter is applied with inputs p2, p1, p0 and q0, producing filtered output p'1 (luma only). If |q2 − q0| is less than the threshold β, a four-tap filter is applied with inputs q2, q1, q0 and p0, producing filtered output q'1 (luma only).

(b) bS = 4

If |p2 − p0| < β and |p0 − q0| < round (α/4) and this is a luma block:

p'0 is produced by five-tap filtering of p2, p1, p0, q0 and q1,
p'1 is produced by four-tap filtering of p2, p1, p0 and q0,
p'2 is produced by five-tap filtering of p3, p2, p1, p0 and q0,

else:

p'0 is produced by three-tap filtering of p1, p0 and q1.

If |q2 − q0| < β and |p0 − q0| < round(α/4) and this is a luma block:

q'0 is produced by five-tap filtering of q2, q1, q0, p0 and p1,
q'1 is produced by four-tap filtering of q2, q1, q0 and p0,
q'2 is produced by five-tap filtering of q3, q2, q1, q0 and p0,

else:

q'0 is produced by three-tap filtering of q1, q0 and p1.

Example

A video clip is encoded with a fixed Quantisation Parameter of 36 (relatively high quantisation). Figure 6.32 shows an original frame from the clip and Figure 6.33 shows the same frame after

inter coding and decoding, with the loop filter disabled. Note the obvious blocking artefacts and note also the effect of varying motion-compensation block sizes (for example, 16 × 16 blocks in the background to the left of the picture, 4 × 4 blocks around the arm). With the loop filter enabled (Figure 6.34) there is still some obvious distortion but most of the block edges have disappeared or faded. Note that sharp contrast boundaries (such as the line of the arm against the dark piano) are preserved by the filter whilst block edges in smoother regions of the picture (such as the background to the left) are smoothed. In this example the loop filter makes only a small contribution to compression efficiency: the encoded bitrate is around 1.5% smaller and the PSNR around 1% larger for the sequence with the filter. However, the subjective quality of the filtered sequence is significantly better. The coding performance gain provided by the filter depends on the bitrate and sequence content.

Figure 6.35 and Figure 6.36 show the un-filtered and filtered frame respectively, this time with a lower quantiser parameter ($QP = 32$).

6.4.8 Transform and Quantisation

H.264 uses three transforms depending on the type of residual data that is to be coded: a Hadamard transform for the 4 × 4 array of luma DC coefficients in intra macroblocks predicted in 16 × 16 mode, a Hadamard transform for the 2 × 2 array of chroma DC coefficients (in any macroblock) and a DCT-based transform for all other 4 × 4 blocks in the residual data.

Data within a macroblock are transmitted in the order shown in Figure 6.47. If the macroblock is coded in 16 × 16 Intra mode, then the block labelled '−1', containing the transformed DC coefficient of each 4 × 4 luma block, is transmitted first. Next, the luma residual blocks 0–15 are transmitted in the order shown (the DC coefficients in a macroblock coded in 16 × 16 Intra mode are not sent). Blocks 16 and 17 are sent, containing a 2 × 2 array of DC coefficients from the Cb and Cr chroma components respectively and finally, chroma residual blocks 18–25 (without DC coefficients) are sent.

6.4.8.1 4 × 4 Residual Transform and Quantisation (blocks 0–15, 18–25)

This transform operates on 4 × 4 blocks of residual data (labelled 0–15 and 18–25 in Figure 6.37) after motion-compensated prediction or Intra prediction. The H.264 transform [3] is based on the DCT but with some fundamental differences:

1. It is an integer transform (all operations can be carried out using integer arithmetic, without loss of decoding accuracy).
2. It is possible to ensure zero mismatch between encoder and decoder inverse transforms (using integer arithmetic).
3. The core part of the transform can be implemented using only additions and shifts.
4. A scaling multiplication (part of the transform) is integrated into the quantiser, reducing the total number of multiplications.

The inverse quantisation (scaling) and inverse transform operations can be carried out using 16-bit integer arithmetic (footnote: except in the case of certain anomalous residual data patterns) with only a single multiply per coefficient, without any loss of accuracy.

Figure 6.32 Original frame (violin frame 2)

Figure 6.33 Reconstructed, QP = 36 (no filter)

Figure 6.34 Reconstructed, QP = 36 (with filter)

Figure 6.35 Reconstructed, QP = 32 (no filter)

Figure 6.36 Reconstructed, QP = 32 (with filter)

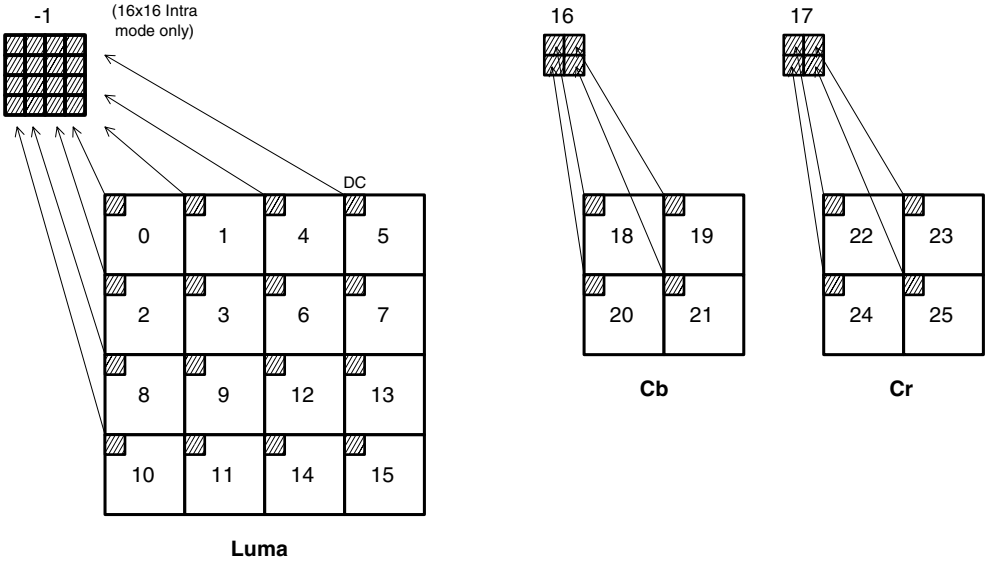


Figure 6.37 Scanning order of residual blocks within a macroblock

Development from the 4 × 4 DCT

Recall from Chapter 3 that the 4 × 4 DCT is given by:

$$\mathbf{Y} = \mathbf{AXA}^{\mathrm{T}} = \begin{bmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & -c \end{bmatrix} \begin{bmatrix} & \mathbf{X} & \end{bmatrix} \begin{bmatrix} a & b & a & c \\ a & c & -a & -b \\ a & -c & -a & b \\ a & -b & a & -c \end{bmatrix} \tag{6.1}$$

where:

$$a = \frac{1}{2}, \qquad b = \sqrt{\frac{1}{2}}\cos\left(\frac{\pi}{8}\right), \qquad c = \sqrt{\frac{1}{2}}\cos\left(\frac{3\pi}{8}\right)$$

This matrix multiplication can be factorised [3] to the following equivalent form (equation 6.2):

$$\boldsymbol{Y} = (CXC^{\mathrm{T}}) \otimes \boldsymbol{E} = \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & d & -d & -1 \\ 1 & -1 & -1 & 1 \\ d & -1 & 1 & -d \end{bmatrix} \begin{bmatrix} & \mathbf{X} & \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & d \\ 1 & d & -1 & -1 \\ 1 & -d & -1 & 1 \\ 1 & -1 & 1 & -d \end{bmatrix} \right) \otimes \begin{bmatrix} a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \\ a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \end{bmatrix} \tag{6.2}$$

$\mathbf{CXC}^{\mathrm{T}}$ is a 'core' 2D transform. $\mathbf{E}$ is a matrix of scaling factors and the symbol $\otimes$ indicates that each element of $(\mathbf{CXC}^{\mathrm{T}})$ is multiplied by the scaling factor in the same position in matrix $\mathbf{E}$ (scalar multiplication rather than matrix multiplication). The constants a and b are as before and d is c/b (approximately 0.414).

To simplify the implementation of the transform, d is approximated by 0.5. In order to ensure that the transform remains orthogonal, b also needs to be modified so that:

$$a = \frac{1}{2}, \qquad b = \sqrt{\frac{2}{5}}, \qquad d = \frac{1}{2}$$

The 2nd and 4th rows of matrix $\mathbf{C}$ and the 2nd and 4th columns of matrix $\mathbf{C}^{\mathrm{T}}$ are scaled by a factor of two and the post-scaling matrix $\mathbf{E}$ is scaled down to compensate, avoiding multiplications by half in the 'core' transform $\mathbf{CXC}^{\mathrm{T}}$ which could result in loss of accuracy using integer arithmetic. The final forward transform becomes:

$$\boldsymbol{Y} = \boldsymbol{C}_f \boldsymbol{X} \boldsymbol{C}_f^{\mathrm{T}} \otimes \boldsymbol{E}_f = \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \begin{bmatrix} & \mathbf{X} & \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{bmatrix} \right) \otimes \begin{bmatrix} a^2 & \frac{ab}{2} & a^2 & \frac{ab}{2} \\ \frac{ab}{2} & \frac{b^2}{4} & \frac{ab}{2} & \frac{b^2}{4} \\ a^2 & \frac{ab}{2} & a^2 & \frac{ab}{2} \\ \frac{ab}{2} & \frac{b^2}{4} & \frac{ab}{2} & \frac{b^2}{4} \end{bmatrix} \tag{6.3}$$

This transform is an approximation to the 4 × 4 DCT but because of the change to factors d and b, the result of the new transform will not be identical to the 4 × 4 DCT.

Example

Compare the output of the 4 × 4 approximate transform with the output of a 'true' 4 × 4 DCT, for input block $\mathbf{X}$:

$\mathbf{X}$:

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

DCT output:

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

Approximate transform output:

$$\mathbf{Y}' = (\mathbf{CXC}^{\mathbf{T}}) \otimes \mathbf{E}_f = \begin{bmatrix} 35.0 & -0.158 & -1.5 & 1.107 \\ -3.004 & -3.900 & 1.107 & -9.200 \\ 5.5 & 2.688 & 2.0 & 4.901 \\ -4.269 & -3.200 & -9.329 & -2.100 \end{bmatrix}$$

Difference between DCT and integer transform outputs:

$$\mathbf{Y} - \mathbf{Y}' = \begin{bmatrix} 0 & 0.079 & 0 & 0.008 \\ -0.295 & -0.868 & -0.664 & 0.190 \\ 0 & 0.341 & 0 & -0.203 \\ 0.224 & 0.190 & -0.055 & 0.868 \end{bmatrix}$$

There is clearly a difference in the output coefficients that depend on b or d. In the context of the H.264 CODEC, the approximate transform has almost identical compression performance to the DCT and has a number of important advantages. The 'core' part of the transform, $\mathbf{CXC}^{\mathrm{T}}$, can be carried out with integer arithmetic using only additions, subtractions and shifts. The dynamic range of the transform operations is such that 16-bit arithmetic may be used throughout (except in the case of certain anomalous input patterns) since the inputs are in the range ± 255. The post-scaling operation $\otimes \mathbf{E}_f$ requires one multiplication for every coefficient which can be 'absorbed' into the quantisation process (see below).

The inverse transform is given by Equation 6.4. The H.264 standard [1] defines this transform explicitly as a sequence of arithmetic operations:

$$\boldsymbol{Y} = \boldsymbol{C}_i^T(\boldsymbol{Y} \otimes \boldsymbol{E}_i)\boldsymbol{C}_i = \begin{bmatrix} 1 & 1 & 1 & \frac{1}{2} \\ 1 & \frac{1}{2} & -1 & -1 \\ 1 & -\frac{1}{2} & -1 & 1 \\ 1 & -1 & 1 & -\frac{1}{2} \end{bmatrix} \left(\begin{bmatrix} & \mathbf{X} & \end{bmatrix} \otimes \begin{bmatrix} a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \\ a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \end{bmatrix} \right) \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \frac{1}{2} & -\frac{1}{2} & -1 \\ 1 & -1 & -1 & 1 \\ \frac{1}{2} & -1 & 1 & -\frac{1}{2} \end{bmatrix} \tag{6.4}$$

This time, $\mathbf{Y}$ is *pre-scaled* by multiplying each coefficient by the appropriate weighting factor from matrix $\mathbf{E}_i$. Note the factors $\pm 1/2$ in the matrices $\mathbf{C}$ and $\mathbf{C}^{\mathrm{T}}$ which can be implemented by a right-shift without a significant loss of accuracy because the coefficients $\mathbf{Y}$ are pre-scaled.

The forward and inverse transforms are orthogonal, i.e. $\mathbf{T}^{-1}(\mathbf{T}(\mathbf{X})) = \mathbf{X}$.

Quantisation

H.264 assumes a scalar quantiser (see Chapter 3). The mechanisms of the forward and inverse quantisers are complicated by the requirements to (a) avoid division and/or floating point arithmetic and (b) incorporate the post- and pre-scaling matrices $\mathbf{E}_f$ and $\mathbf{E}_i$ described above.

The basic forward quantiser [3] operation is:

$$Z_{ij} = round(Y_{ij}/Qstep)$$

where Y_{ij} is a coefficient of the transform described above, Qstep is a quantizer step size and Z_{ij} is a quantised coefficient. The rounding operation here (and throughout this section) need not round to the nearest integer; for example, biasing the 'round' operation towards smaller integers can give perceptual quality improvements.

A total of 52 values of Qstep are supported by the standard, indexed by a Quantisation Parameter, QP (Table 6.5). Qstep doubles in size for every increment of 6 in QP. The wide range of quantiser step sizes makes it possible for an encoder to control the tradeoff accurately and flexibly between bit rate and quality. The values of QP can be different for luma and chroma. Both parameters are in the range 0–51 and the default is that the chroma parameter

Table 6.5 Quantisation step sizes in H.264 CODEC

QP	0	1	2	3	4	5	6	7	8	9	10	11	12	...
QStep	0.625	0.6875	0.8125	0.875	1	1.125	1.25	1.375	1.625	1.75	2	2.25	2.5	...
QP	...	18	...	24	...	30	...	36	...	42	...	48	...	51
QStep		5		10		20		40		80		160		224

QP_C is derived from QP_Y so that QP_C is less that QP_Y for $QP_Y > 30$. A user-defined mapping between QP_Y and QP_C may be signalled in a Picture Parameter Set.

The post-scaling factor a^2, $ab/2$ or $b^2/4$ (equation 6.3) is incorporated into the forward quantiser. First, the input block $\mathbf{X}$ is transformed to give a block of unscaled coefficients $\mathbf{W} = \mathbf{CXC}^T$. Then, each coefficient W_{ij} is quantised and scaled in a single operation:

$$Z_{ij} = round\left(W_{ij}.\frac{PF}{Qstep}\right) \tag{6.5}$$

PF is a^2, $ab/2$ or $b^2/4$ depending on the position (i, j) (see equation 6.3):

Position	PF
(0,0), (2,0), (0,2) or (2,2)	a^2
(1,1), (1,3), (3,1) or (3,3)	$b^2/4$
other	ab/2

In order to simplify the arithmetic), the factor ($PF/Qstep$) is implemented in the reference model software [4] as a multiplication by a factor MF and a right-shift, avoiding any division operations:

$$Z_{ij} = round\left(W_{ij}.\frac{MF}{2^{qbits}}\right)$$

where

$$\frac{MF}{2^{qbits}} = \frac{PF}{Qstep}$$

and

$$qbits = 15 + floor(QP/6) \tag{6.6}$$

In integer arithmetic, Equation 6.6 can be implemented as follows:

$$\begin{aligned} |Z_{ij}| &= (|W_{ij}|.MF + f) >> qbits \\ sign(Z_{ij}) &= sign(W_{ij}) \end{aligned} \tag{6.7}$$

where $>>$ indicates a binary shift right. In the reference model software, f is $2^{qbits}/3$ for Intra blocks or $2^{qbits}/6$ for Inter blocks.

Table 6.6 Multiplication factor MF

QP	Positions (0,0),(2,0),(2,2),(0,2)	Positions (1,1),(1,3),(3,1),(3,3)	Other positions
0	13107	5243	8066
1	11916	4660	7490
2	10082	4194	6554
3	9362	3647	5825
4	8192	3355	5243
5	7282	2893	4559

Example

$QP = 4$ and $(i, j) = (0,0)$.

$Qstep = 1.0$, $PF = a^2 = 0.25$ and $qbits = 15$, hence $2^{qbits} = 32768$.

$$\frac{MF}{2^{qbits}} = \frac{PF}{Qstep}, \qquad MF = (32768 \times 0.25)/1 = 8192$$

The first six values of MF (for each coefficient position) used by the H.264 reference software encoder are given in Table 6.6. The 2nd and 3rd columns of this table (positions with factors $b^2/4$ and $ab/2$) have been modified slightly[4] from the results of equation 6.6.
For $QP > 5$, the factors MF remain unchanged but the divisor 2^{qbits} increases by a factor of two for each increment of six in QP. For example, $qbits = 16$ for $6 \leq$ QP ≤ 11, $qbits = 17$ for $12 \leq$QP≤ 17 and so on.

ReScaling

The basic scaling (or 'inverse quantiser') operation is:

$$Y'_{ij} = Z_{ij} Qstep \tag{6.8}$$

The pre-scaling factor for the inverse transform (from matrix $\mathbf{E}_i$, containing values a^2, ab and b^2 depending on the coefficient position) is incorporated in this operation, together with a constant scaling factor of 64 to avoid rounding errors:

$$W'_{ij} = Z_{ij} Qstep \cdot PF \cdot 64 \tag{6.9}$$

W'_{ij} is a scaled coefficient which is transformed by the core inverse transform $\mathbf{C}_i^T \mathbf{W} \mathbf{C}_i$ (Equation 6.4). The values at the output of the inverse transform are divided by 64 to remove the scaling factor (this can be implemented using only an addition and a right-shift). The H.264 standard does not specify Qstep or PF directly. Instead, the parameter $V = (Qstep.PF.64)$ is defined for $0 \leq QP \leq 5$ and for each coefficient position so that the scaling

[4] It is acceptable to modify a forward quantiser, for example in order to improve perceptual quality at the decoder, since only the rescaling (inverse quantiser) process is standardised.

operation becomes:

$$W'_{ij} = Z_{ij} V_{ij} \cdot 2^{floor(QP/6)} \tag{6.10}$$

Example

$QP = 3$ and $(i, j) = (1, 2)$
$Qstep = 0.875$ and $2^{\text{floor(QP/6)}} = 1$
$PF = ab = 0.3162$
$V = (Qstep \cdot PF \cdot 64) = 0.875 \times 0.3162 \times 65 \cong 18$
$W'_{ij} = Z_{ij} \times 18 \times 1$

The values of V defined in the standard for $0 \leq QP \leq 5$ are shown in Table 6.7.

The factor $2^{\text{floor}(QP/6)}$ in Equation 6.10 causes the sclaed output increase by a factor of two for every increment of six in QP.

6.4.9 4 × 4 Luma DC Coefficient Transform and Quantisation (16 × 16 Intra-mode Only)

If the macroblock is encoded in 16 × 16 Intra prediction mode (i.e. the entire 16 × 16 luma component is predicted from neighbouring samples), each 4 × 4 residual block is first transformed using the 'core' transform described above ($\mathbf{C}_f \mathbf{X} \mathbf{C}_f^T$). The DC coefficient of each 4 × 4 block is then transformed again using a 4 × 4 Hadamard transform:

$$\mathbf{Y}_D = \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} & \mathbf{W}_D & \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \right) /2 \tag{6.11}$$

$\mathbf{W}_D$ is the block of 4 × 4 DC coefficients and $\mathbf{Y}_D$ is the block after transformation. The output coefficients $Y_{D(i,j)}$ are quantised to produce a block of quantised DC coefficients:

$$\begin{aligned} \left|Z_{D(i,j)}\right| &= \left(\left|Y_{D(i,j)}\right| MF_{(0,0)} + 2f\right) >> (qbits + 1) \\ \text{sign}\left(Z_{D(i,j)}\right) &= \text{sign}\left(Y_{D(i,j)}\right) \end{aligned} \tag{6.12}$$

$MF_{(0,0)}$ is the multiplication factor for position (0,0) in Table 6.6 and f, $qbits$ are defined as before.

At the decoder, an inverse Hadamard transform is applied *followed by* rescaling (note that the order is not reversed as might be expected):

$$\mathbf{W}_{QD} = \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} & \mathbf{Z}_D & \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \right) \tag{6.13}$$

Table 6.7 Scaling factor V

QP	Positions (0,0),(2,0),(2,2),(0,2)	Positions (1,1),(1,3),(3,1),(3,3)	Other positions
0	10	16	13
1	11	18	14
2	13	20	16
3	14	23	18
4	16	25	20
5	18	29	23

Decoder scaling is performed by:

$$W'_{D(i,j)} = W_{QD(i,j)}V_{(0,0)}2^{\text{floor}}(QP/6) - 2 \qquad (QP \geq 12)$$
$$W'_{D(i,j)} = \left[W_{QD(i,j)}V_{(0,0)} + 2^{1-floor(QP/6)}\right] >> (2 - floor(QP/6) \qquad (QP < 12) \tag{6.14}$$

$V_{(0,0)}$ is the scaling factor V for position (0,0) in Table 6.7. Because $V_{(0,0)}$ is constant throughout the block, rescaling and inverse transformation can be applied in any order. The specified order (inverse transform first, then scaling) is designed to maximise the dynamic range of the inverse transform.

The rescaled DC coefficients $\mathbf{W}'_D$ are inserted into their respective 4×4 blocks and each 4×4 block of coefficients is inverse transformed using the core DCT-based inverse transform ($\mathbf{C}_i^T\mathbf{W}'\mathbf{C}_i$). In a 16×16 intra-coded macroblock, much of the energy is concentrated in the DC coefficients of each 4×4 block which tend to be highly correlated. After this extra transform, the energy is concentrated further into a small number of significant coefficients.

6.4.10 2 × 2 Chroma DC Coefficient Transform and Quantisation

Each 4×4 block in the chroma components is transformed as described in Section 6.4.8.1. The DC coefficients of each 4×4 block of chroma coefficients are grouped in a 2×2 block ($\mathbf{W}_D$) and are further transformed prior to quantisation:

$$\mathbf{W}_{QD} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} \mathbf{W}_D \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{6.15}$$

Quantisation of the 2×2 output block $\mathbf{Y}_D$ is performed by:

$$\left|Z_{D(i,j)}\right| = \left(\left|Y_{D(i,j)}\right| .MF_{(0,0)} + 2f\right) >> (qbits + 1) \tag{6.16}$$
$$\text{sign}\left(Z_{D(i,j)}\right) = \text{sign}\left(Y_{D(i,j)}\right)$$

$MF_{(0,0)}$ is the multiplication factor for position (0,0) in Table 6.6, f and $qbits$ are defined as before.

During decoding, the inverse transform is applied before scaling:

$$\mathbf{W}_{QD} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} \mathbf{Z}_D \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{6.17}$$

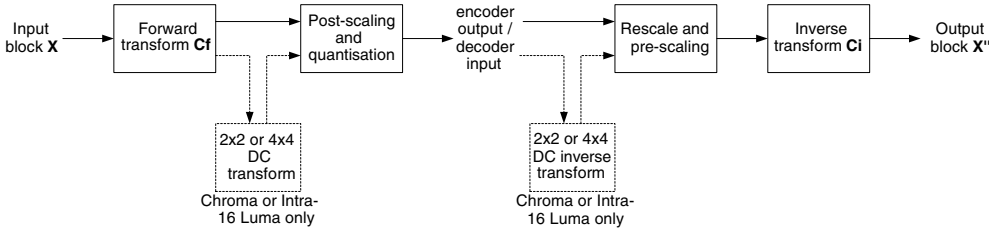


Figure 6.38 Transform, quantisation, rescale and inverse transform flow diagram

Scaling is performed by:

$$W'_{D(i,j)} = W_{QD(i,j)}.V_{(0.0)}.2^{floor(QP/6)-1} \qquad (\text{if } QP \geq 6)$$
$$W'_{D(i,j)} = \left[W_{QD(i,j)}.V_{(0,0)}\right] >> 1 \qquad (\text{if } QP < 6)$$

The rescaled coefficients are replaced in their respective 4×4 blocks of chroma coefficients which are then transformed as above ($\mathbf{C}_i^T\mathbf{W}'\mathbf{C}_i$). As with the Intra luma DC coefficients, the extra transform helps to de-correlate the 2×2 chroma DC coefficients and improves compression performance.

6.4.11 The Complete Transform, Quantisation, Rescaling and Inverse Transform Process

The complete process from input residual block $\mathbf{X}$ to output residual block $\mathbf{X}'$ is described below and illustrated in Figure 6.38.

Encoding:

1. Input: 4×4 residual samples: $\mathbf{X}$
2. Forward 'core' transform: $\mathbf{W} = \mathbf{C}_f\mathbf{X}\mathbf{C}_f^{\mathrm{T}}$
   (followed by forward transform for Chroma DC or Intra-16 Luma DC coefficients).
3. Post-scaling and quantisation: Z = W.round(PF/Qstep)
   (different for Chroma DC or Intra-16 Luma DC).

Decoding:
(Inverse transform for Chroma DC or Intra-16 Luma DC coefficients)

4. Decoder scaling (incorporating inverse transform pre-scaling): $\mathbf{W}' = \mathbf{Z}.Qstep.PF.64$
   (different for Chroma DC or Intra-16 Luma DC).
5. Inverse 'core' transform: $\mathbf{X}' = \mathbf{C}_i^T\mathbf{W}'\mathbf{C}_i$
6. Post-scaling: $\mathbf{X}'' = \text{round}(X'/64)$
7. Output: 4×4 residual samples: $\mathbf{X}''$

Example (luma 4 × 4 residual block, Intra mode)

$QP = 10$

Input block **X:**

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

Output of 'core' transform **W**:

	$j=0$	1	2	3
$i=0$	140	−1	−6	7
1	−19	−39	7	−92
2	22	17	8	31
3	−27	−32	−59	−21

$MF = 8192$, 3355 or 5243 (depending on the coefficient position), $qbits = 16$ and f is $2^{qbits}/3$. Output of forward quantizer **Z**:

	$j=0$	1	2	3
$i=0$	17	0	−1	0
1	−1	−2	0	−5
2	3	1	1	2
3	−2	−1	−5	−1

$V = 16$, 25 or 20 (depending on position) and $2^{\text{floor}}(\text{QP}/6) = 2^1 = 2$. Output of rescale $\mathbf{W}'$:

	$j=0$	1	2	3
$i=0$	544	0	−32	0
1	−40	−100	0	−250
2	96	40	32	80
3	−80	−50	−200	−50

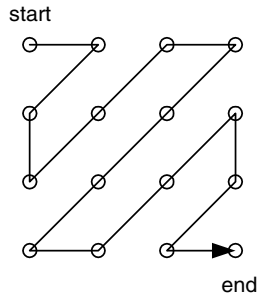


Figure 6.39 Zig-zag scan for 4 × 4 luma block (frame mode)

Output of 'core' inverse transform $\mathbf{X}''$ (after division by 64 and rounding)**:**

	$j=0$	1	2	3
$i=0$	4	13	8	10
1	8	8	4	12
2	1	10	10	3
3	18	5	14	7

6.4.12 Reordering

In the encoder, each 4 × 4 block of quantised transform coefficients is mapped to a 16-element array in a zig-zag order (Figure 6.39). In a macroblock encoded in 16 × 16 Intra mode, the DC coefficients (top-left) of each 4 × 4 luminance block are scanned first and these DC coefficients form a 4 × 4 array that is scanned in the order of Figure 6.39. This leaves 15 AC coefficients in each luma block that are scanned starting from the 2nd position in Figure 6.39. Similarly, the 2 × 2 DC coefficients of each chroma component are first scanned (in raster order) and then the 15 AC coefficients in each chroma 4 × 4 block are scanned starting from the 2nd position.

6.4.13 Entropy Coding

Above the slice layer, syntax elements are encoded as fixed- or variable-length binary codes. At the slice layer and below, elements are coded using either variable-length codes (VLCs) or context-adaptive arithmetic coding (CABAC) depending on the entropy encoding mode. When entropy_coding_mode is set to 0, residual block data is coded using a context-adaptive variable length coding (CAVLC) scheme and other variable-length coded units are coded using Exp-Golomb codes. Parameters that require to be encoded and transmitted include the following (Table 6.8).

Table 6.8 Examples of parameters to be encoded

Parameters	Description
Sequence-, picture- and slice-layer syntax elements	Headers and parameters
Macroblock type mb_type	Prediction method for each coded macroblock
Coded block pattern	Indicates which blocks within a macroblock contain coded coefficients
Quantiser parameter	Transmitted as a delta value from the previous value of QP
Reference frame index	Identify reference frame(s) for inter prediction
Motion vector	Transmitted as a difference (mvd) from predicted motion vector
Residual data	Coefficient data for each 4×4 or 2×2 block

Table 6.9 Exp-Golomb codewords

code_num	Codeword
0	1
1	010
2	011
3	00100
4	00101
5	00110
6	00111
7	0001000
8	0001001
...	...

6.4.13.1 Exp-Golomb Entropy Coding

Exp-Golomb codes (Exponential Golomb codes, [5]) are variable length codes with a regular construction. It is clear from examining the first few codewords (Table 6.9) that they are constructed in a logical way:

[M zeros][1][INFO]

INFO is an M-bit field carrying information. The first codeword has no leading zero or trailing INFO. Codewords 1 and 2 have a single-bit INFO field, codewords 3–6 have a two-bit INFO field and so on. The length of each Exp-Golomb codeword is $(2M + 1)$ bits and each codeword can be constructed by the encoder based on its index *code_num*:

$$\mathrm{M} = \mathrm{floor}(\log_2[\mathrm{code\_num} + 1])$$
$$\mathrm{INFO} = \mathrm{code\_num} + 1 - 2^{\mathrm{M}}$$

A codeword can be decoded as follows:

1. Read in M leading zeros followed by 1.
2. Read M-bit INFO field.
3. code_num = 2^{M}+ INFO – 1

(For codeword 0, INFO and M are zero.)

A parameter k to be encoded is mapped to code_num in one of the following ways:

Mapping type	Description
ue	Unsigned direct mapping, code_num = k. Used for macroblock type, reference frame index and others.
te	A version of the Exp-Golomb codeword table in which short codewords are truncated.
se	Signed mapping, used for motion vector difference, delta QP and others. k is mapped to code_num as follows (Table 6.10). code_num = 2|k| (k ≤ 0) code_num = 2|k| − 1 (k > 0)
me	Mapped symbols, parameter k is mapped to code_num according to a table specified in the standard. Table 6.11 lists a small part of the coded_block_pattern table for Inter predicted macroblocks, indicating which 8 × 8 blocks in a macroblock contain nonzero coefficients.

Table 6.10 Signed mapping se

k	code_num
0	0
1	1
−1	2
2	3
−2	4
3	5
...	...

Table 6.11 Part of coded_block_pattern table

coded_block_pattern (Inter prediction)	code_num
0 (no nonzero blocks)	0
16 (chroma DC block nonzero)	1
1 (top-left 8 × 8 luma block nonzero)	2
2 (top-right 8 × 8 luma block nonzero)	3
4 (lower-left 8 × 8 luma block nonzero)	4
8 (lower-right 8 × 8 luma block nonzero)	5
32 (chroma DC and AC blocks nonzero)	6
3 (top-left and top-right 8 × 8 luma blocks nonzero)	7
...	...

Each of these mappings (ue, te, se and me) is designed to produce short codewords for frequently-occurring values and longer codewords for less common parameter values. For example, inter macroblock type P_L0_16 × 16 (prediction of 16 × 16 luma partition from a previous picture) is assigned code_num 0 because it occurs frequently; macroblock type P_8 × 8 (prediction of 8 × 8 luma partition from a previous picture) is assigned code_num 3 because it occurs less frequently; the commonly-occurring motion vector difference (MVD) value of 0 maps to code_num 0 whereas the less-common MVD = −3 maps to code_num 6.

6.4.13.2 Context-Based Adaptive Variable Length Coding (CAVLC)

This is the method used to encode residual, zig-zag ordered 4 × 4 (and 2 × 2) blocks of transform coefficients. CAVLC [6] is designed to take advantage of several characteristics of quantised 4 × 4 blocks:

1. After prediction, transformation and quantisation, blocks are typically sparse (containing mostly zeros). CAVLC uses run-level coding to represent strings of zeros compactly.
2. The highest nonzero coefficients after the zig-zag scan are often sequences of ±1 and CAVLC signals the number of high-frequency ±1 coefficients ('Trailing Ones') in a compact way.
3. The number of nonzero coefficients in neighbouring blocks is correlated. The number of coefficients is encoded using a look-up table and the choice of look-up table depends on the number of nonzero coefficients in neighbouring blocks.
4. The level (magnitude) of nonzero coefficients tends to be larger at the start of the reordered array (near the DC coefficient) and smaller towards the higher frequencies. CAVLC takes advantage of this by adapting the choice of VLC look-up table for the level parameter depending on recently-coded level magnitudes.

CAVLC encoding of a block of transform coefficients proceeds as follows:

coeff_token	encodes the number of non-zero coefficients (TotalCoeff) and TrailingOnes (one per block)
trailing_ones_sign_flag	sign of TrailingOne value (one per trailing one)
level_prefix	first part of code for non-zero coefficient (one per coefficient, excluding trailing ones)
level_suffix	second part of code for non-zero coefficient (not always present)
total_zeros	encodes the total number of zeros occurring after the first non-zero coefficient (in zig-zag order) (one per block)
run_before	encodes number of zeros preceding each non-zero coefficient *in reverse zig-zag order*

1. Encode the number of coefficients and trailing ones (coeff_token)

The first VLC, coeff_token, encodes both the total number of nonzero coefficients (TotalCoeffs) and the number of trailing ±1 values (TrailingOnes). TotalCoeffs can be anything from 0 (no coefficients in the 4 × 4 block)[5] to 16 (16 nonzero coefficients) and TrailingOnes can be anything from 0 to 3. If there are more than three trailing ±1s, only the last three are treated as 'special cases' and any others are coded as normal coefficients.

There are four choices of look-up table to use for encoding coeff_token for a 4 × 4 block, three variable-length code tables and a fixed-length code table. The choice of table depends on the number of nonzero coefficients in the left-hand and upper previously coded blocks (n_A and n_B respectively). A parameter nC is calculated as follows. If upper and left blocks nB and nA

[5] Note: coded_block_pattern (described earlier) indicates which *8 × 8* blocks in the macroblock contain nonzero coefficients but, within a coded 8 × 8 block, there may be *4 × 4* sub-blocks that do not contain any coefficients, hence TotalCoeff may be 0 in any 4 × 4 sub-block. In fact, this value of TotalCoeff occurs most often and is assigned the shortest VLC.

Table 6.12 Choice of look-up table for coeff_token

N	Table for coeff_token
0, 1	Table 1
2, 3	Table 2
4, 5, 6, 7	Table 3
8 or above	Table 4

are both available (i.e. in the same coded slice), nC = round((nA + nB)/2). If only the upper is available, nC = nB; if only the left block is available, nC = nA; if neither is available, nC = 0.

The parameter nC selects the look-up table (Table 6.12) so that the choice of VLC adapts to the number of coded coefficients in neighbouring blocks (*context adaptive*). Table 1 is biased towards small numbers of coefficients such that low values of TotalCoeffs are assigned particularly short codes and high values of TotalCoeff particularly long codes. Table 2 is biased towards medium numbers of coefficients (TotalCoeff values around 2–4 are assigned relatively short codes), Table 3 is biased towards higher numbers of coefficients and Table 4 assigns a fixed six-bit code to every pair of TotalCoeff and TrailingOnes values.

2. Encode the sign of each TrailingOne

For each TrailingOne (trailing ±1) signalled by coeff_token, the sign is encoded with a single bit (0 = +, 1 = −) *in reverse order*, starting with the highest-frequency TrailingOne.

3. Encode the levels of the remaining nonzero coefficients.

The *level* (sign and magnitude) of each remaining nonzero coefficient in the block is encoded *in reverse order*, starting with the highest frequency and working back towards the DC coefficient. The code for each level is made up of a prefix (level_prefix) and a suffix (level_suffix). The length of the suffix (suffixLength) may be between 0 and 6 bits and suffixLength is adapted depending on the magnitude of each successive coded level ('context adaptive'). A small value of suffixLength is appropriate for levels with low magnitudes and a larger value of suffixLength is appropriate for levels with high magnitudes. The choice of suffixLength is adapted as follows:

1. Initialise suffixLength to 0 (unless there are more than 10 nonzero coefficients and less than three trailing ones, in which case initialise to 1).
2. Encode the highest-frequency nonzero coefficient.
3. If the magnitude of this coefficient is larger than a predefined threshold, increment suffixLength. (If this is the first level to be encoded and suffixLength was initialised to 0, set suffixLength to 2).

In this way, the choice of suffix (and hence the complete VLC) is matched to the magnitude of the recently-encoded coefficients. The thresholds are listed in Table 6.13; the first threshold is

Table 6.13 Thresholds for determining whether to increment suffixLength

Current suffixLength	Threshold to increment suffixLength
0	0
1	3
2	6
3	12
4	24
5	48
6	N/A (highest suffixLength)

zero which means that suffixLength is always incremented after the first coefficient level has been encoded.

4. Encode the total number of zeros before the last coefficient
The sum of all zeros preceding the highest nonzero coefficient in the reordered array is coded with a VLC, total zeros. The reason for sending a separate VLC to indicate total zeros is that many blocks contain a number of nonzero coefficients at the start of the array and (as will be seen later) this approach means that zero-runs at the start of the array need not be encoded.

5. Encode each run of zeros.
The number of zeros preceding each nonzero coefficient (run_before) is encoded *in reverse order*. A run_before parameter is encoded for each nonzero coefficient, starting with the highest frequency, with two exceptions:

1. If there are no more zeros left to encode (i.e. $\sum$[run_before] = total_zeros), it is not necessary to encode any more run_before values.
2. It is not necessary to encode run_before for the final (lowest frequency) nonzero coefficient.

The VLC for each run of zeros is chosen depending on (a) the number of zeros that have not yet been encoded (ZerosLeft) and (b) run_before. For example, if there are only two zeros left to encode, run_before can only take three values (0, 1 or 2) and so the VLC need not be more than two bits long. If there are six zeros still to encode then run_before can take seven values (0 to 6) and the VLC table needs to be correspondingly larger.

Example 1

4 × 4 block:

0	3	−1	0
0	−1	1	0
1	0	0	0
0	0	0	0

Reordered block:
0,3,0,1,−1, −1,0,1,0. . .

TotalCoeffs = 5 (indexed from highest frequency, 4, to lowest frequency, 0)
total_zeros = 3
TrailingOnes = 3 (in fact there are four trailing ones but only three can be encoded as a 'special case')

Encoding

Element	Value	Code
coeff_token	TotalCoeffs = 5, TrailingOnes= 3 (use Table 1)	0000100
TrailingOne sign (4)	+	0
TrailingOne sign (3)	−	1
TrailingOne sign (2)	−	1
Level (1)	+1 (use suffixLength = 0)	1 (prefix)
Level (0)	+3 (use suffixLength = 1)	001 (prefix) 0 (suffix)
total zeros	3	111
run_before(4)	ZerosLeft = 3; run_before = 1	10
run_before(3)	ZerosLeft = 2; run_before = 0	1
run_before(2)	ZerosLeft = 2; run_before = 0	1
run_before(1)	ZerosLeft = 2; run_before = 1	01
run_before(0)	ZerosLeft = 1; run_before = 1	No code required; last coefficient.

The transmitted bitstream for this block is 000010001110010111101101.

Decoding

The output array is 'built up' from the decoded values as shown below. Values added to the output array at each stage are underlined.

Code	Element	Value	Output array
0000100	coeff_token	TotalCoeffs = 5, TrailingOnes = 3	Empty
0	TrailingOne sign	+	$\underline{1}$
1	TrailingOne sign	−	$\underline{-1}$, 1
1	TrailingOne sign	−	$\underline{-1}$, −1, 1
1	Level	+1 (suffixLength = 0; increment suffixLength after decoding)	$\underline{1}$, −1, −1, 1
0010	Level	+3 (suffixLength = 1)	$\underline{3}$, 1, −1, −1, 0, 1
111	total_zeros	3	3, 1, −1, −1, 1
10	run_before	1	3, 1, −1, −1, $\underline{0}$, 1
1	run_before	0	3, 1, −1, −1, 0, 1
1	run_before	0	3, 1, −1, −1, 0, 1
01	run_before	1	3, $\underline{0}$, 1, −1, −1, 0, 1

The decoder has already inserted two zeros, TotalZeros is equal to 3 and so another 1 zero is inserted before the lowest coefficient, making the final output array:

$$\underline{0}, 3, 0, 1, -1, -1, 0, 1$$

Example 2

4 × 4 block:

−2	4	0	−1
3	0	0	0
−3	0	0	0
0	0	0	0

Reordered block:
−2, 4, 3, −3, 0, 0, −1, . . .

TotalCoeffs = 5 (indexed from highest frequency, 4, to lowest frequency, 0)
total_zeros = 2
TrailingOne = 1

Encoding:

Element	Value	Code
coeff_token	TotalCoeffs = 5, TrailingOnes = 1 (use Table 1)	0000000110
TrailingOne sign (4)	−	1
Level (3)	Sent as −2 (*see note* 1) (suffixLength = 0; increment suffixLength)	0001 (prefix)
Level (2)	3 (suffixLength = 1)	001 (prefix) 0 (suffix)
Level (1)	4 (suffixLength = 1; increment suffixLength	0001 (prefix) 0 (suffix)
Level (0)	−2 (suffixLength = 2)	1 (prefix) 11 (suffix)
total zeros	2	0011
run_before(4)	ZerosLeft= 2; run_before= 2	00
run_before(3..0)	0	No code required

The transmitted bitstream for this block is 000000011010001001000010111001100.

Note 1: Level (3), with a value of −3, is encoded as a special case. If there are less than 3 TrailingOnes, then the first *non*-trailing one level cannot have a value of ±1 (otherwise it would have been encoded as a TrailingOne). To save bits, this level is incremented if negative (decremented if positive) so that ±2 maps to ±1, ±3 maps to ±2, and so on. In this way, shorter VLCs are used.

Note 2: After encoding level (3), the level_VLC table is incremented because the magnitude of this level is greater than the first threshold (which is 0). After encoding level (1), with a magnitude of 4, the table number is incremented again because level (1) is greater than the second threshold (which is 3). Note that the final level (−2) uses a different VLC from the first encoded level (also –2).

Decoding:

Code	Element	Value	Output array
0000000110	coeff_token	TotalCoeffs = 5, T1s= 1	Empty
1	TrailingOne sign	−	−1
0001	Level	−2 decoded as −3	−3, −1
0010	Level	+3	+3, −3, −1
00010	Level	+4	+4, 3, −3, −1
111	Level	−2	−2, 4, 3, −3, −1
0011	total_zeros	2	−2, 4, 3, −3, −1
00	run_before	2	−2, 4, 3, −3, 0, 0, −1

All zeros have now been decoded and so the output array is:
−2, 4, 3, −3, 0, 0, −1
(This example illustrates how bits are saved by encoding TotalZeros: only a single zero run (run_before) needs to be coded even though there are five nonzero coefficients.)

Example 3

4 × 4 block:

0	0	1	0
0	0	0	0
1	0	0	0
−0	0	0	0

Reordered block:
0,0,0,1,0,1,0,0,0,−1

TotalCoeffs = 3 (indexed from highest frequency [2] to lowest frequency [0])
total_zeros = 7
TrailingOnes = 3

Encoding:

Element	Value	Code
coeff_token	TotalCoeffs = 3, TrailingOnes= 3 use Table 1)	00011
TrailingOne sign (2)	−	1
TrailingOne sign (1)	+	0
TrailingOne sign (0)	+	0
total_zeros	7	011
run_before(2)	ZerosLeft= 7; run_before= 3	100
run_before(1)	ZerosLeft= 4; run_before= 1	10
run_before(0)	ZerosLeft= 3; run_before= 3	No code required; last coefficient.

The transmitted bitstream for this block is 0001110001110010.

Decoding:

Code	Element	Value	Output array
00011	coeff_token	TotalCoeffs= 3, TrailingOnes= 3	Empty
1	TrailineOne sign	−	−1
0	TrailineOne sign	+	1, −1
0	TrailineOne sign	+	1, 1, −1
011	total_zeros	7	1, 1, −1
100	run_before	3	1, 1, 0, 0, 0, −1
10	run_before	1	1, 0, 1, 0, 0, 0, −1

The decoder has inserted four zeros. total_zeros is equal to 7 and so another three zeros are inserted before the lowest coefficient:

$$0, 0, 0, 1, 0, 1, 0, 0, 0, -1$$

6.5 THE MAIN PROFILE

Suitable application for the Main Profile include (but are not limited to) broadcast media applications such as digital television and stored digital video. The Main Profile is almost a superset of the Baseline Profile, except that multiple slice groups, ASO and redundant slices (all included in the Baseline Profile) are not supported. The additional tools provided by Main Profile are B slices (bi-predicted slices for greater coding efficiency), weighted prediction (providing increased flexibility in creating a motion-compensated prediction block), support for interlaced video (coding of fields as well as frames) and CABAC (an alternative entropy coding method based on Arithmetic Coding).

6.5.1 B slices

Each macroblock partition in an inter coded macroblock in a B slice may be predicted from one or two reference pictures, before or after the current picture in temporal order. Depending on the reference pictures stored in the encoder and decoder (see the next section), this gives many options for choosing the prediction references for macroblock partitions in a B macroblock type. Figure 6.40 shows three examples: (a) one past and one future reference (similar to B-picture prediction in earlier MPEG video standards), (b) two past references and (c) two future references.

6.5.1.1 Reference pictures

B slices use two lists of previously-coded reference pictures, list 0 and list 1, containing short term and long term pictures (see Section 6.4.2). These two lists can each contain past and/or

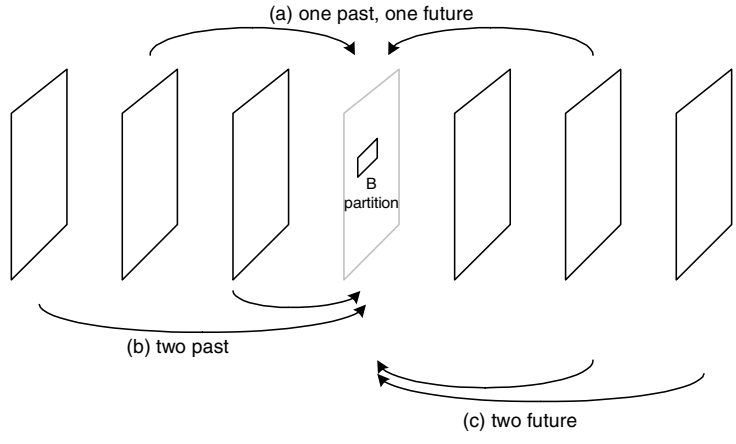


Figure 6.40 Partition prediction examples in a B macroblock type: (a) past/future, (b) past, (c) future

future coded pictures (pictures before or after the current picture in display order). The long term pictures in each list behaves in a similar way to the description in Section 6.4.2. The short term pictures may be past and/or future coded pictures and the default index order of these pictures is as follows:

List 0: The closest past picture (based on picture order count) is assigned index 0, followed by any other past pictures (increasing in picture order count), followed by any future pictures (in increasing picture order count from the current picture).

List 1: The closest future picture is assigned index 0, followed by any other future picture (in increasing picture order count), followed by any past picture (in increasing picture order count).

Example

An H.264 decoder stores six short term reference pictures with picture order counts: 123, 125, 126, 128, 129, 130. The current picture is 127. All six short term reference pictures are marked as used for reference in list 0 and list 1. The pictures are indexed in the list 0 and list 1 short term buffers as follows (Table 6.14).

Table 6.14 Short term buffer indices (B slice prediction) (current picture order count is 127

Index	List 0	List 1
0	126	128
1	125	129
2	123	130
3	128	126
4	129	125
5	130	123

Table 6.15 Prediction options in B slice macroblocks

Partition	Options
16 × 16	Direct, list 0, list1 or bi-predictive
16 × 8 or 8 × 16	List 0, list 1 or bi-predictive (chosen separately for each partition)
8 × 8	Direct, list 0, list 1 or bi-predictive (chosen separately for each partition).

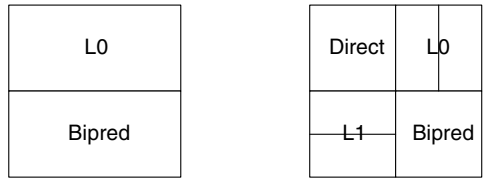


Figure 6.41 Examples of prediction modes in B slice macroblocks

The selected buffer index is sent as an Exp-Golomb codeword (see Section 6.4.13.1) and so the most efficient choice of reference index (with the smallest codeword) is index 0 (i.e. the previous coded picture in list 0 and the next coded picture in list 1).

6.5.1.2 Prediction Options

Macroblocks partitions in a B slice may be predicted in one of several ways, direct mode (see Section 6.5.1.4), motion-compensated prediction from a list 0 reference picture, motion-compensated prediction from a list 1 reference picture, or motion-compensated bi-predictive prediction from list 0 and list 1 reference pictures (see Section 6.5.1.3). Different prediction modes may be chosen for each partition (Table 6.15); if the 8 × 8 partition size is used, the chosen mode for each 8 × 8 partition is applied to all sub-partitions within that partition. Figure 6.41 shows two examples of valid prediction mode combinations. On the left, two 16 × 8 partitions use List 0 and Bi-predictive prediction respectively and on the right, four 8 × 8 partitions use Direct, List 0, List 1 and Bi-predictive prediction.

6.5.1.3 Bi-prediction

In Bi-predictive mode, a reference block (of the same size as the current partition or sub-macroblock partition) is created from the list 0 and list 1 reference pictures. Two motion-compensated reference areas are obtained from a list 0 and a list 1 picture respectively (and hence two motion vectors are required) and each sample of the prediction block is calculated as an average of the list 0 and list 1 prediction samples. Except when using Weighted Prediction (see Section 6.5.2), the following equation is used:

$$\text{pred}(i,j) = (\text{pred0}(i,j) + \text{pred1}(i,j) + 1) >> 1$$

Pred0(i, j) and pred1(i, j) are prediction samples derived from the list 0 and list 1 reference frames and pred(i, j) is a bi-predictive sample. After calculating each prediction sample, the motion-compensated residual is formed by subtracting pred(i, j) from each sample of the current macroblock as usual.

Example

A macroblock is predicted in B_Bi_16 × 16 mode (i.e. bi-prediction of the complete macroblock). Figure 6.42 and Figure 6.43 show motion-compensated reference areas from list 0 and list 1 references pictures respectively and Figure 6.44 shows the bi-prediction formed from these two reference areas.

The list 0 and list 1 vectors in a bi-predictive macroblock or block are each predicted from neighbouring motion vectors that have the same temporal direction. For example a vector for the current macroblock pointing to a past frame is predicted from other neighbouring vectors that also point to past frames.

6.5.1.4 Direct Prediction

No motion vector is transmitted for a B slice macroblock or macroblock partition encoded in Direct mode. Instead, the decoder calculates list 0 and list 1 vectors based on previously-coded vectors and uses these to carry out bi-predictive motion compensation of the decoded residual samples. A skipped macroblock in a B slice is reconstructed at the decoder using Direct prediction.

A flag in the slice header indicates whether a spatial or temporal method will be used to calculate the vectors for direct mode macroblocks or partitions.

In *spatial direct* mode, list 0 and list 1 predicted vectors are calculated as follows. Predicted list 0 and list 1 vectors are calculated using the process described in section 6.4.5.3. If the co-located MB or partition in the first list 1 reference picture has a motion vector that is less than $\pm 1/2$ luma samples in magnitude (and in some other cases), one or both of the predicted vectors are set to zero; otherwise the predicted list 0 and list 1 vectors are used to carry out bi-predictive motion compensation. In *temporal direct* mode, the decoder carries out the following steps:

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

These modes are modified when, for example, the prediction reference macroblocks or partitions are not available or are intra coded.

Example:

The list 1 reference for the current macroblock occurs two pictures after the current frame (Figure 6.45). The co-located MB in the list 1 reference has a vector MV(+2.5, +5) pointing to a list 0 reference picture that occurs three pictures before the current picture. The decoder calculates MV1(−1, −2) and MV0(+1.5, +3) pointing to the list 1 and list 0 pictures respectively. These vectors are derived from MV and have magnitudes proportional to the picture order count distance to the list 0 and list 1 reference frames.

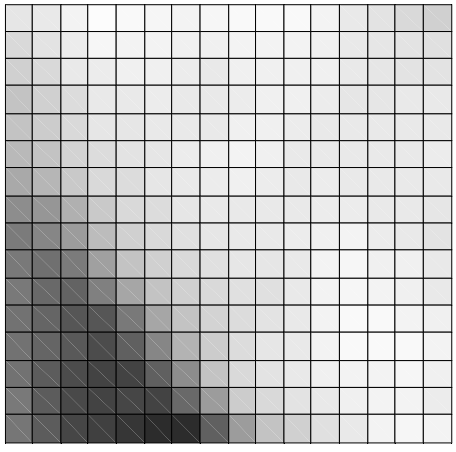


Figure 6.42 Reference area (list 0 picture)

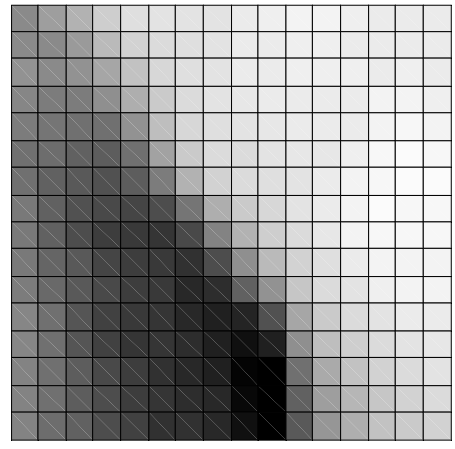


Figure 6.43 Reference area (list 1 picture)

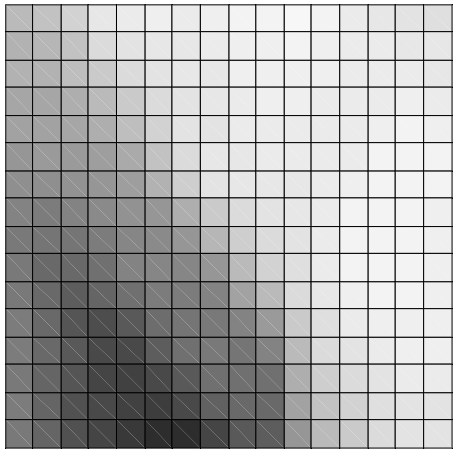


Figure 6.44 Prediction (non-weighted)

6.5.2 Weighted Prediction

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

1. P slice macroblock, 'explicit' weighted prediction;
2. B slice macroblock, 'explicit' weighted prediction;
3. B slice macroblock, 'implicit' weighted prediction.

Each prediction sample pred0(i, j) or pred1(i, j) is scaled by a weighting factor w_0 or w_1 prior to motion-compensated prediction. In the 'explicit' types, the weighting factor(s) are

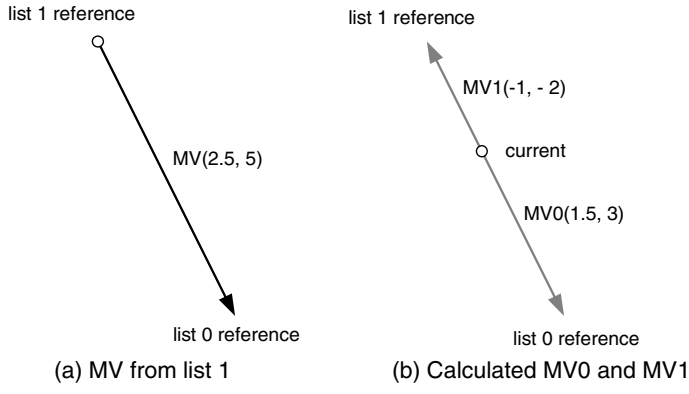


Figure 6.45 Temporal direct motion vector example

determined by the encoder and transmitted in the slice header. If 'implicit' prediction is used, w_0 and w_1 are calculated based on the relative temporal positions of the list 0 and list 1 reference pictures. A larger weighting factor is applied if the reference picture is temporally close to the current picture and a smaller factor is applied if the reference picture is temporally further away from the current picture.

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

6.5.3 Interlaced Video

Efficient coding of interlaced video requires tools that are optimised for compression of field macroblocks. If field coding is supported, the type of picture (frame or field) is signalled in the header of each slice. In macroblock-adaptive frame/field (**MB-AFF**) coding mode, the choice of field or frame coding may be specified at the macroblock level. In this mode, the current slice is processed in units of 16 luminance samples wide and 32 luminance samples high, each of which is coded as a 'macroblock pair' (Figure 6.46). The encoder can choose to encode each MB pair as (a) two frame macroblocks or (b) two field macroblocks and may select the optimum coding mode for each region of the picture.

Coding a slice or MB pair in field mode requires modifications to a number of the encoding and decoding steps described in Section 6.4. For example, each coded field is treated as a separate reference picture for the purposes of P and B slice prediction, the prediction of coding modes in intra MBs and motion vectors in inter MBs require to be modified depending on whether adjacent MBs are coded in frame or field mode and the reordering scan shown in Figure 6.47 replaces the zig-zag scan of Figure 6.39.

6.5.4 Context-based Adaptive Binary Arithmetic Coding (CABAC)

When the picture parameter set flag entropy_coding_mode is set to 1, an arithmetic coding system is used to encode and decode H.264 syntax elements. Context-based Adaptive Binary

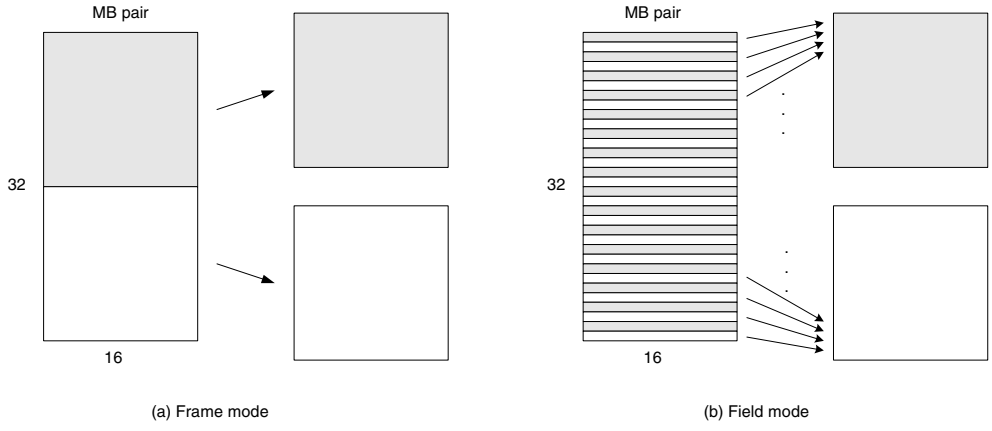


Figure 6.46 Macroblock-adaptive frame/field coding

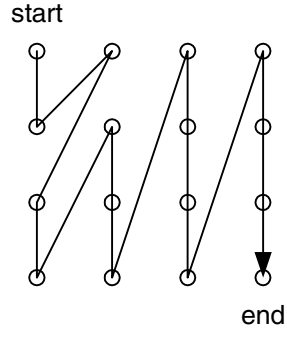


Figure 6.47 Reordering scan for 4 × 4 luma blocks (field mode)

Arithmetic Coding (CABAC) [7], achieves good compression performance through (a) selecting probability models for each syntax element according to the element's context, (b) adapting probability estimates based on local statistics and (c) using arithmetic coding rather than variable-length coding. Coding a data symbol involves the following stages:

1. Binarisation: CABAC uses Binary Arithmetic Coding which means that only binary decisions (1 or 0) are encoded. A non-binary-valued symbol (e.g. a transform coefficient or motion vector, any symbol with more than 2 possible values) is 'binarised' or converted into a binary code prior to arithmetic coding. This process is similar to the process of converting a data symbol into a variable length code (Section 6.4.13) but the binary code is further encoded (by the arithmetic coder) prior to transmission.

Stages 2, 3 and 4 are repeated for each bit (or 'bin') of the binarised symbol:

2. Context model selection. A 'context model' is a probability model for one or more bins of the binarised symbol and is chosen from a selection of available models depending on the statistics of recently-coded data symbols. The context model stores the probability of each bin being '1' or '0'.

3. Arithmetic encoding: An arithmetic coder encodes each bin according to the selected probability model (see section 3.5.3). Note that there are just two sub-ranges for each bin (corresponding to '0' and '1').
4. Probability update: The selected context model is updated based on the actual coded value (e.g. if the bin value was '1', the frequency count of '1's is increased).

The Coding Process

We will illustrate the coding process for one example, mvd_x (motion vector difference in the x-direction, coded for each partition or sub-macroblock partition in an inter macroblock).

1. Binarise the value $\text{mvd}_x \cdots \text{mvd}_x$ is mapped to the following table of uniquely-decodeable codewords for $|\text{mvd}_x| < 9$ (larger values of mvd_x are binarised using an Exp-Golomb codeword).

$\lvert\text{mvd}_x\rvert$	Binarisation (s=sign)
0	0
1	10s
2	110s
3	1110s
4	11110s
5	111110s
6	1111110s
7	11111110s
8	111111110s

   The first bit of the binarised codeword is bin 1, the second bit is bin 2 and so on.
2. Choose a *context model* for each bin. One of three models is selected for bin 1 (Table 6.16), based on the L1 norm of two previously-coded mvd_x values, e_k:

$$e_k = |\text{mvd}_{\text{xA}}| + |\text{mvd}_{\text{xB}}| \quad \text{where A and B are the blocks immediately to the left and above the current block.}$$

   If e_k is small, then there is a high probability that the current MVD will have a small magnitude and, conversely, if e_k is large then it is more likely that the current MVD will have a large magnitude. A probability table (context model) is selected accordingly. The remaining bins are coded using one of four further context models (Table 6.17).

Table 6.16 context models for bin 1

e_k	Context model for bin 1
$0 \leq e_k < 3$	Model 0
$3 \leq e_k < 33$	Model 1
$33 \leq e_k$	Model 2

Table 6.17 Context models

Bin	Context model
1	0, 1 or 2 depending on e_k
2	3
3	4
4	5
5 and higher	6
6	6

3. Encode each bin. The selected context model supplies two probability estimates, the probability that the bin contains '1' and the probability that the bin contains '0', that determine the two sub-ranges used by the arithmetic coder to encode the bin.
4. Update the context models. For example, if context model 2 is selected for bin 1 and the value of bin 1 is '0', the frequency count of '0's is incremented so that the next time this model is selected, the probability of an '0' will be slightly higher. When the total number of occurrences of a model exceeds a threshold value, the frequency counts for '0' and '1' will be scaled down, which in effect gives higher priority to recent observations.

The Context Models

Context models and binarisation schemes for each syntax element are defined in the standard. There are nearly 400 separate context models for the various syntax elements. At the beginning of each coded slice, the context models are initialised depending on the initial value of the Quantisation Parameter QP (since this has a significant effect on the probability of occurrence of the various data symbols). In addition, for coded P, SP and B slices, the encoder may choose one of 3 sets of context model initialisation parameters at the beginning of each slice, to allow adaptation to different types of video content [8].

The Arithmetic Coding Engine

The arithmetic decoder is described in some detail in the Standard and has three distinct properties:

1. Probability estimation is performed by a transition process between 64 separate probability states for 'Least Probable Symbol' (LPS, the least probable of the two binary decisions '0' or '1').
2. The range R representing the current state of the arithmetic coder (see Chapter 3) is quantised to a small range of pre-set values before calculating the new range at each step, making it possible to calculate the new range using a look-up table (i.e. multiplication-free).
3. A simplified encoding and decoding process (in which the context modelling part is bypassed) is defined for data symbols with a near-uniform probability distribution.

The definition of the decoding process is designed to facilitate low-complexity implementations of arithmetic encoding and decoding. Overall, CABAC provides improved coding efficiency compared with VLC (see Chapter 7 for performance examples).

6.6 THE EXTENDED PROFILE

The Extended Profile (known as the X Profile in earlier versions of the draft H.264 standard) may be particularly useful for applications such as video streaming. It includes all of the features of the Baseline Profile (i.e. it is a superset of the Baseline Profile, unlike Main Profile), together with B-slices (Section 6.5.1), Weighted Prediction (Section 6.5.2) and additional features to support efficient streaming over networks such as the Internet. SP and SI slices facilitate switching between different coded streams and 'VCR-like' functionality and Data Partitioned slices can provide improved performance in error-prone transmission environments.

6.6.1 SP and SI slices

SP and SI slices are specially-coded slices that enable (among other things) efficient switching between video streams and efficient random access for video decoders [10]. A common requirement in a streaming application is for a video decoder to switch between one of several encoded streams. For example, the same video material is coded at multiple bitrates for transmission across the Internet and a decoder attempts to decode the highest-bitrate stream it can receive but may require switching automatically to a lower-bitrate stream if the data throughput drops.

Example

A decoder is decoding Stream A and wants to switch to decoding Stream B (Figure 6.48). For simplicity, assume that each frame is encoded as a single slice and predicted from one reference (the previous decoded frame). After decoding P-slices A_0 and A_1, the decoder wants to switch to Stream B and decode B_2, B_3 and so on. If all the slices in Stream B are coded as P-slices, then the decoder will not have the correct decoded reference frame(s) required to reconstruct B_2 (since B_2 is predicted from the decoded picture B_1 which does not exist in stream A). One solution is to code frame B_2 as an I-slice. Because it is coded without prediction from any other frame, it can be decoded independently of preceding frames in stream B and the decoder can therefore switch between stream A and stream B as shown in Figure 6.49. Switching can be accommodated by inserting an I-slice at regular intervals in the coded sequence to create 'switching points'. However, an I-slice is likely to contain much more coded data than a P-slice and the result is an undesirable peak in the coded bitrate at each switching point.

SP-slices are designed to support switching between similar coded sequences (for example, the same source sequence encoded at various bitrates) without the increased bitrate penalty of I-slices (Figure 6.49). At the switching point (frame 2 in each sequence), there are three SP-slices, each coded using motion compensated prediction (making them more efficient than I-slices). SP-slice A_2 can be decoded using reference picture A_1 and SP-slice B_2 can be decoded using reference picture B_1. The key to the switching process is SP-slice AB_2 (known as a *switching SP-slice*), created in such a way that it can be decoded using motion-compensated reference picture A_1, to produce decoded frame B_2 (i.e. the decoder output frame B_2 is identical whether decoding B_1 followed by B_2 or A_1 followed by AB_2). An extra SP-slice is required at each switching point (and in fact another SP-slice, BA_2, would be required to switch in the other direction) but this is likely to be more efficient than encoding frames A_2

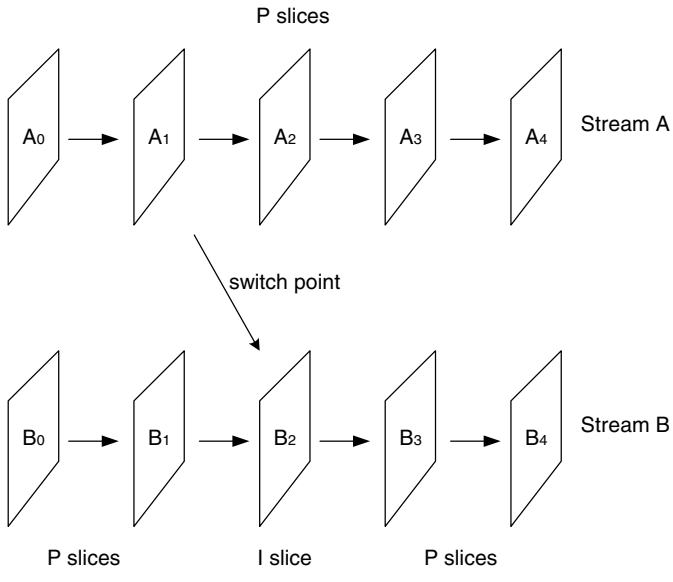


Figure 6.48 Switching streams using I-slices

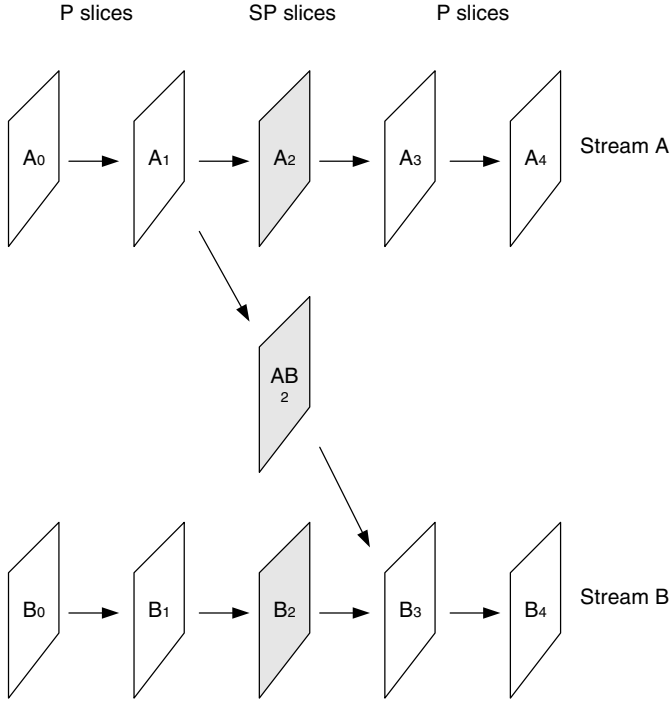


Figure 6.49 Switching streams using SP-slices

Table 6.18 Switching from stream A to stream B using SP-slices

Input to decoder	MC reference	Output of decoder
P-slice A_0	[earlier frame]	Decoded frame A_0
P-slice A_1	Decoded frame A_0	Decoded frame A_1
SP-slice AB_2	Decoded frame A_1	Decoded frame B_2
P-slice B_3	Decoded frame B_2	Decoded frame B_3
....	....	....

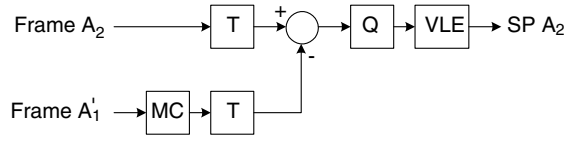


Figure 6.50 Encoding SP-slice A_2 (simplified)

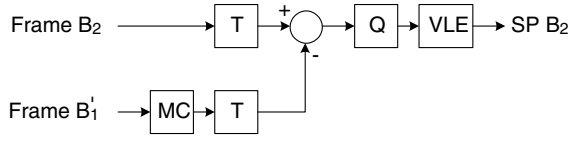


Figure 6.51 Encoding SP-slice B_2 (simplified)

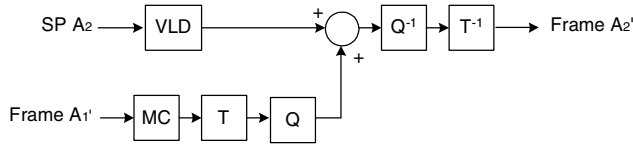


Figure 6.52 Decoding SP-slice A_2 (simplified)

and B_2 as I-slices. Table 6.18 lists the steps involved when a decoder switches from stream A to stream B.

Figure 6.50 shows a simplified diagram of the encoding process for SP-slice A_2, produced by subtracting a motion-compensated version of A_1' (decoded frame A_1) from frame A_2 and then coding the residual. Unlike a 'normal' P-slice, the subtraction occurs in the transform domain (after the block transform). SP-slice B_2 is encoded in the same way (Figure 6.51). A decoder that has previously decoded frame A_1 can decode SP-slice A_2 as shown in Figure 6.52. Note that these diagrams are simplified; in practice further quantisation and rescaling steps are required to avoid mismatch between encoder and decoder and a more detailed treatment of the process can be found in [11].

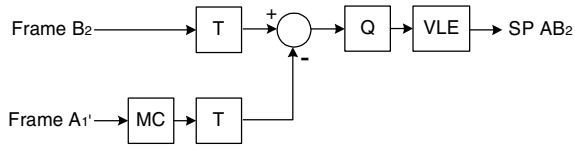


Figure 6.53 Encoding SP-slice AB_2 (simplified)

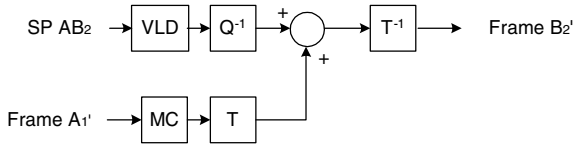


Figure 6.54 Decoding SP-slice AB_2 (simplified)

SP-slice AB_2 is encoded as shown in Figure 6.53 (simplified). Frame B_2 (the frame we are switching to) is transformed and a motion-compensated prediction is formed from A_1' (the decoded frame from which we are switching). The 'MC' block in this diagram attempts to find the best match for each MB of frame B_2 *using decoded picture* A_1 *as a reference*. The motion-compensated prediction is transformed, then subtracted from the transformed B_2 (i.e. in the case of a switching SP slice, subtraction takes place in the transform domain). The residual (after subtraction) is quantized, encoded and transmitted.

A decoder that has previously decoded A_1' can decode SP-slice AB_2 to produce B_2' (Figure 6.54). A_1' is motion compensated (using the motion vector data encoded as part of AB_2), transformed and added to the decoded and scaled (inverse quantized) residual, then the result is inverse transformed to produce B_2'.

If streams A and B are versions of the same original sequence coded at different bitrates, the motion-compensated prediction of B_2 from A_1' (SP-slice AB_2) should be quite efficient. Results show that using SP-slices to switch between different versions of the same sequence is significantly more efficient than inserting I-slices at switching points. Another application of SP-slices is to provide random access and 'VCR-like' functionalities. For example, an SP-slice and a switching SP-slice are placed at the position of frame 10 (Figure 6.55). A decoder can fast-forward from A_0 directly to A_{10} by (a) decoding A_0, then (b) decoding switching SP-slice A_{0-10} to produce A_{10} by prediction from A_0.

A further type of switching slice, the SI-slice, is supported by the Extended Profile. This is used in a similar way to a switching SP-slice, except that the prediction is formed using the 4×4 Intra Prediction modes (see Section 6.4.6.1) from previously-decoded samples of the reconstructed frame. This slice mode may be used (for example) to switch from one sequence to a completely different sequence (in which case it will not be efficient to use motion compensated prediction because there is no correlation between the two sequences).

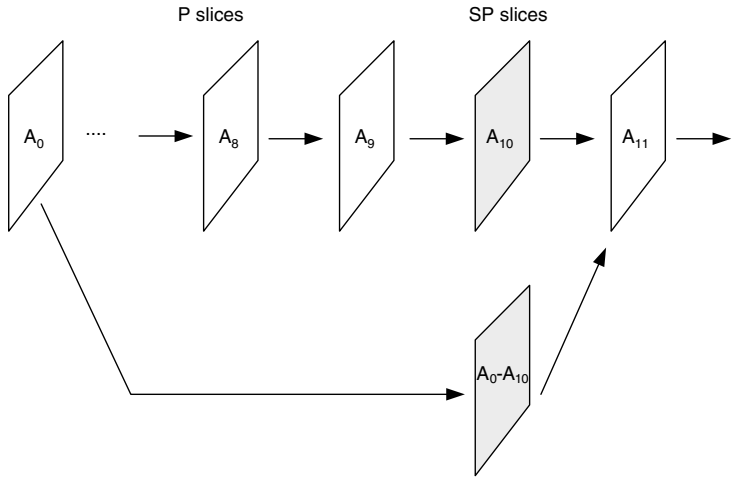


Figure 6.55 Fast-forward using SP-slices

Sequence parameter set	SEI	Picture parameter set	I slice	Picture delimiter	P slice	P slice	.....

Figure 6.56 Example sequence of RBSP elements

6.6.2 Data Partitioned Slices

The coded data that makes up a slice is placed in three separate Data Partitions (A, B and C), each containing a subset of the coded slice. Partition A contains the slice header and header data for each macroblock in the slice, Partition B contains coded residual data for Intra and SI slice macroblocks and Partition C contains coded residual data for inter coded macroblocks (forward and bi-directional). Each Partition can be placed in a separate NAL unit and may therefore be transported separately.

If Partition A data is lost, it is likely to be difficult or impossible to reconstruct the slice, hence Partition A is highly sensitive to transmission errors. Partitions B and C can (with careful choice of coding parameters) be made to be independently decodeable and so a decoder may (for example) decode A and B only, or A and C only, lending flexibility in an error-prone environment.

6.7 TRANSPORT OF H.264

A coded H.264 video sequence consists of a series of NAL units, each containing an RBSP (Table 6.19). Coded slices (including Data Partitioned slices and IDR slices) and the End of Sequence RBSP are defined as VCL NAL units whilst all other elements are just NAL units.

An example of a typical sequence of RBSP units is shown in Figure 6.56. Each of these units is transmitted in a separate NAL unit. The header of the NAL unit (one byte) signals the type of RBSP unit and the RBSP data makes up the rest of the NAL unit.

Table 6.19

RBSP type	Description
Parameter Set	'Global' parameters for a sequence such as picture dimensions, video format, macroblock allocation map (see Section 6.4.3).
Supplemental Enhancement Information	Side messages that are not essential for correct decoding of the video sequence.
Picture Delimiter	Boundary between video pictures (optional). If not present, the decoder infers the boundary based on the frame number contained within each slice header.
Coded slice	Header and data for a slice; this RBSP unit contains actual coded video data.
Data Partition A, B or C	Three units containing Data Partitioned slice layer data (useful for error resilient decoding). Partition A contains header data for all MBs in the slice, Partition B contains intra coded data and partition C contains inter coded data.
End of sequence	Indicates that the next picture (in decoding order) is an IDR picture (see Section 6.4.2). (Not essential for correct decoding of the sequence).
End of stream	Indicates that there are no further pictures in the bitstream. (Not essential for correct decoding of the sequence).
Filler data	Contains 'dummy' data (may be used to increase the number of bytes in the sequence). (Not essential for correct decoding of the sequence).

Parameter sets

H.264 introduces the concept of *parameter sets*, each containing information that can be applied to a large number of coded pictures. A *sequence parameter set* contains parameters to be applied to a complete video sequence (a set of consecutive coded pictures). Parameters in the sequence parameter set include an identifier (seq_parameter_set_id), limits on frame numbers and picture order count, the number of reference frames that may be used in decoding (including short and long term reference frames), the decoded picture width and height and the choice of progressive or interlaced (frame or frame / field) coding. A *picture parameter set* contains parameters which are applied to one or more decoded pictures within a sequence. Each picture parameter set includes (among other parameters) an identifier (pic_parameter_set_id), a selected seq_parameter_set_id, a flag to select VLC or CABAC entropy coding, the number of slice groups in use (and a definition of the type of slice group map), the number of reference pictures in list 0 and list 1 that may be used for prediction, initial quantizer parameters and a flag indicating whether the default deblocking filter parameters are to be modified.

Typically, one or more sequence parameter set(s) and picture parameter set(s) are sent to the decoder prior to decoding of slice headers and slice data. A coded slice header refers to a pic_parameter_set_id and this 'activates' that particular picture parameter set. The 'activated' picture parameter set then remains active until a different picture parameter set is activated by being referred to in another slice header. In a similar way, a picture parameter set refers to a seq_parameter_set_id which 'activates' that sequence parameter set. The activated set remains in force (i.e. its parameters are applied to all consecutive coded pictures) until a different sequence parameter set is activated.

The parameter set mechanism enables an encoder to signal important, infrequently-changing sequence and picture parameters separately from the coded slices themselves. The parameter sets may be sent well ahead of the slices that refer to them, or by another transport

mechanism (e.g. over a reliable transmission channel or even 'hard wired' in a decoder implementation). Each coded slice may 'call up' the relevant picture and sequence parameters using a single VLC (pic_parameter_set_id) in the slice header.

Transmission and Storage of NAL units

The method of transmitting NAL units is not specified in the standard but some distinction is made between transmission over packet-based transport mechanisms (e.g. packet networks) and transmission in a continuous data stream (e.g. circuit-switched channels). In a packet-based network, each NAL unit may be carried in a separate packet and should be organised into the correct sequence prior to decoding. In a circuit-switched transport environment, a start code prefix (a uniquely-identifiable delimiter code) is placed before each NAL unit to make a *byte stream* prior to transmission. This enables a decoder to search the stream to find a start code prefix identifying the start of a NAL unit.

In a typical application, coded video is required to be transmitted or stored together with associated audio track(s) and side information. It is possible to use a range of transport mechanisms to achieve this, such as the Real Time Protocol and User Datagram Protocol (RTP/UDP). An Amendment to MPEG-2 Systems specifies a mechanism for transporting H.264 video (see Chapter 7) and ITU-T Recommendation H.241 defines procedures for using H.264 in conjunction with H.32× multimedia terminals. Many applications require storage of multiplexed video, audio and side information (e.g. streaming media playback, DVD playback). A forthcoming Amendment to MPEG-4 Systems (Part 1) specifies how H.264 coded data and associated media streams can be stored in the ISO Media File Format (see Chapter 7).

6.8 CONCLUSIONS

H.264 provides mechanisms for coding video that are optimised for compression efficiency and aim to meet the needs of practical multimedia communication applications. The range of available coding tools is more restricted than MPEG-4 Visual (due to the narrower focus of H.264) but there are still many possible choices of coding parameters and strategies. The success of a practical implementation of H.264 (or MPEG-4 Visual) depends on careful design of the CODEC and effective choices of coding parameters. The next chapter examines design issues for each of the main functional blocks of a video CODEC and compares the performance of MPEG-4 Visual and H.264.

6.9 REFERENCES

1. ISO/IEC 14496-10 and ITU-T Rec. H.264, Advanced Video Coding, 2003.
2. T. Wiegand, G. Sullivan, G. Bjontegaard and A. Luthra, Overview of the H.264 / AVC Video Coding Standard, IEEE Transactions on Circuits and Systems for Video Technology, to be published in 2003.
3. A. Hallapuro, M. Karczewicz and H. Malvar, Low Complexity Transform and Quantization – Part I: Basic Implementation, JVT document JVT-B038, Geneva, February 2002.
4. H.264 Reference Software Version JM6.1d, http://bs.hhi.de/~suehring/tml/, March 2003.

5. S. W. Golomb, Run-length encoding, *IEEE* Trans. on Inf. Theory, **IT-12**, pp. 399–401, 1966.
6. G. Bjøntegaard and K. Lillevold, Context-adaptive VLC coding of coefficients, JVT document JVT-C028, Fairfax, May 2002.
7. D. Marpe, G. Blättermann and T. Wiegand, Adaptive codes for H.26L, ITU-T SG16/6 document VCEG-L13, Eibsee, Germany, January 2001.
8. H. Schwarz, D. Marpe and T. Wiegand, CABAC and slices, JVT document JVT-D020, Klagenfurt, Austria, July 2002
9. D. Marpe, H. Schwarz and T. Wiegand, Context-Based Adaptive Binary Arithmetic Coding in the H.264 / AVC Video Compression Standard, IEEE Transactions on Circuits and Systems for Video Technology, to be published in 2003.
10. M. Karczewicz and R. Kurceren, A proposal for SP-frames, ITU-T SG16/6 document VCEG-L27, Eibsee, Germany, January 2001.
11. M. Karczewicz and R. Kurceren, The SP and SI Frames Design for H.264/AVC, IEEE Transactions on Circuits and Systems for Video Technology, to be published in 2003.