New Video Coding Standards: H.264 and H.265

12

12.1 H.264

The Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG (Video Coding Experts Group) developed the H.264 video compression standard. It was formerly known by its working title "H.26L." The final draft of the first version of H.264 was completed in May 2003 [1].

H.264 is also known as MPEG-4 Part 10, AVC (Advanced Video Coding) [2–4]. It is often referred to as the H.264/AVC (or H.264/MPEG-4 AVC) video coding standard.

H.264/AVC provides a higher video coding efficiency, up to 50 % better compression than MPEG-2 and up to 30 % better than H.263+ and MPEG-4 Advanced Simple Profile, while maintaining the same quality of the compressed video. It covers a broad range of applications, from high bitrate to very low bitrate. The vastly improved H.264 core features, together with new coding tools offer significant improvement in compression ratio, error resiliency, and subjective quality over existing ITU-T and MPEG standards. It has since become the default standard for various applications, e.g., the Blu-ray discs, HDTV broadcasts, streaming video on the Internet, web software such as Flash and Silverlight, and apps on mobile and portable devices.

Similar to previous video compression standards, H.264 specifies a block-based, hybrid coding scheme that supports motion compensation and transform coding. Again, each picture can be separated into macroblocks (16×16 blocks), and arbitrary-sized slices can group multiple macroblocks into self-contained units. The basic H.264/AVC encoder is shown in Fig. 12.1.

Main features of H.264/AVC are:

- Integer transform in 4×4 blocks. Low complexity, no drifting.
- Variable block-size motion compensation, from 16×16 to 4×4 in luma images.
- Quarter-pixel accuracy in motion vectors, accomplished by interpolations.
- Multiple reference picture motion compensation. More than just P or B frames for motion estimation.
- Directional spatial prediction for intra frames.

Z.-N. Li et al., *Fundamentals of Multimedia*,
Texts in Computer Science, DOI: 10.1007/978-3-319-05290-8_12,

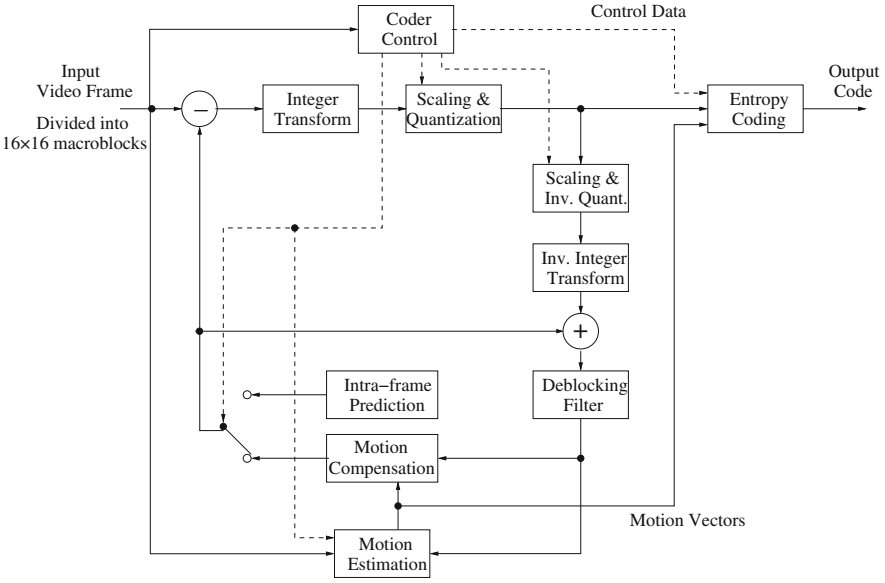

Fig. 12.1 Basic encoder for H.264/AVC

- In-loop deblocking filtering.
- Context-Adaptive Variable Length Coding (CAVLC) and Context-Adaptive Binary Arithmetic Coding (CABAC).
- More robust to data errors and data losses, more flexible in synchronization and switching of video streams produced by different decoders.

The decoder has the following five major blocks:

- Entropy decoding
- Inverse quantization and transform of residual pixels
- Motion compensation or intra-prediction
- Reconstruction
- In-loop deblocking filter on reconstructed pixels.

12.1.1 Motion Compensation

Similar to MPEG-2 and H.263, H.264 employs the technology of *hybrid coding*, i.e., a combination of inter-picture motion predictions and intra-picture spatial prediction, and transform coding on residual errors.

Variable Block-Size Motion Compensation

As before, inter-frame motion estimation in H.264 is also block-based. By default, the macroblocks are of the size 16×16. A macroblock can also be divided into four 8×8 *partitions*. While conducting motion estimation, each macroblock or each of

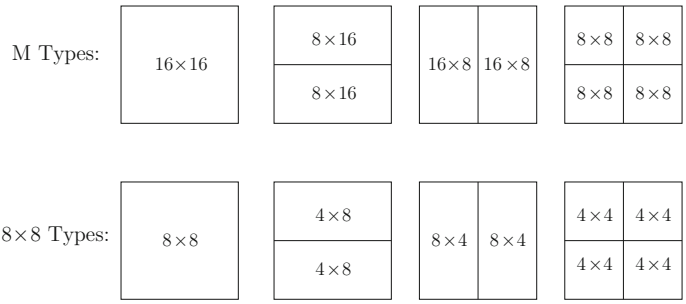

Fig. 12.2 Segmentation of the macroblock for motion estimation in H.264. *Top* Segmentation of the macroblock. *Bottom* Segmentation of the 8 × 8 partition

the partitions can be further segmented into smaller partitions as shown in Fig. 12.2. The top four options are from the 16 × 16 macroblock (the so-called M Types), and the bottom four options are from each of the 8 × 8 partitions (the so-called 8 × 8 Types).

Quarter-Pixel Precision

The accuracy of motion compensation is of quarter-pixel precision in luma images. Fig. 12.3 illustrates how the pixel values at half-pixel and quarter-pixel positions can be derived by interpolation. In order to derive the pixel values at half-pixel positions labeled b and h, the intermediate values b_1 and h_1 are first derived by applying the six-tap filters as below.

$$b_1 = E - 5F + 20G + 20H - 5I + J$$
$$h_1 = A - 5C + 20G + 20M - 5R + T.$$

The values of b and h are then obtained by the following, clipped to the range 0–255.

$$b = (b_1 + 16) \gg 5$$
$$h = (h_1 + 16) \gg 5.$$

The special symbol "≫" in above formulas indicates a right-shift. For example, $b = (b_1 + 16) \gg 5$ is equivalent to $b = \text{round}(b_1/2^5) = \text{round}(b_1/32)$. However, the shift operation is much more efficient compared to a *round* function call.

The middle pixel 'j' is obtained by

$$j_1 = aa_1 - 5bb_1 + 20h_1 + 20m_1 - 5cc_1 + dd_1,$$

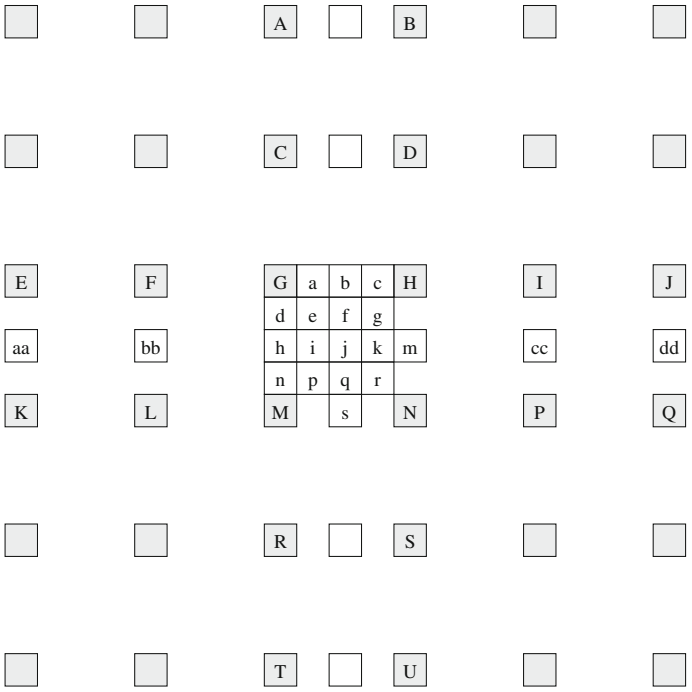

Fig. 12.3 Interpolation for fractional samples in H.264. *Upper-case letters* indicate pixels on the image grid. *Lower-case letters* indicate pixels at half-pixel and quarter-pixel positions

where the intermediate values aa_1, bb_1, m_1, cc_1, and dd_1 are derived in a similar manner as h_1. Then, the value of j is obtained by the following, clipped to the range of 0 to 255.

$$j = (j_1 + 512) \gg 10.$$

The pixel values at quarter-pixel positions labeled a, c, d, n, f, i, k, and q are obtained by averaging the values of the two nearest pixels at integer and half-pixel positions. For example,

$$a = (G + b + 1) \gg 1.$$

Finally, the pixel values at quarter-pixel positions labeled e, g, p, and r are obtained by averaging the values of the two nearest pixels at half-pixel positions in the diagonal direction. For example,

$$e = (b + h + 1) \gg 1.$$

Additional Options in Group of Pictures

As shown in Fig. 11.3, in previous MPEG standards, the *Group of Pictures (GOP)* starts and ends with I-frames. In between it has P-frames and B-frames. Either an I-frame or P-frame can be used as *Reference frame*. Macroblocks in P-frames are predicted by forward prediction, and macroblocks in B-frames are predicted by a combination of forward prediction and backward prediction. H.264 will continue supporting this "classic" GOP structure. In addition, it will support the following GOP structures.

No B-frames
The prediction of macroblocks in B-frames incurs more delay and requires more storage for the necessary I- and P-frames because of the bidirectional prediction. In this option, only I- and P-frames are allowed. Although the compression efficiency is relatively low, it is more suitable for certain applications, e.g., videoconferencing where minimal delay is more desirable. This is compatible with the goals of the Baseline Profile or Constrained Baseline Profile of H.264.

Multiple reference frames
In order to find the best match for each macroblock in P-frames, H.264 allows up to N reference frames. Figure 12.4 illustrates an example where $N = 4$. The reference frame for P_1 is I_0, the reference frames for P_2 are I_0 and $P_1, \ldots$ For P_4, the reference frames are I_0, P_1, P_2, and P_3. While this improves the compression efficiency, it requires much more computation in motion estimation at the encoder. Moreover, it requires a larger buffer to store up to N frames at the encoder and decoder.

Hierarchical prediction structure
Hierarchical prediction structures are also allowed under H.264's flexible prediction options. For example, we can have a GOP that starts and ends with I_0 and I_{12}. In between there are 11 consecutive B-frames, B_1 to B_{11}. First, B_6 is predicted using I_0 and I_{12} as references. Next, B_3 is predicted using I_0 and B_6, and B_9 is predicted using B_6 and I_{12} as references. At last, B_1 and B_2 are predicted using I_0 and B_3, B_4 and B_5 are predicted using B_3 and B_6, and so on. This hierarchical structure can be seen as having layers. For this example, Layer 0: I_0, I_{12}; Layer 1: B_6; Layer 2: B_3, B_9; Layer 3: B_1, B_2, B_4, B_5, B_7, B_8, B_{10}, B_{11}. Usually, increasingly larger quantization parameters will be associated with higher layers to control the compression efficiency. This is shown to be more efficient than the IBBP. . . structure used in previous video coding standards for temporal prediction.

12.1.2 Integer Transform

As in previous video coding standards, H.264 employs Transform coding after difference macroblocks are obtained. One of the most important features in H.264/AVC is the use of an *Integer Transform*.

The Discrete Cosine Transform (DCT) in previous video coding standards is known to cause *prediction shift* because of floating point calculation and rounding

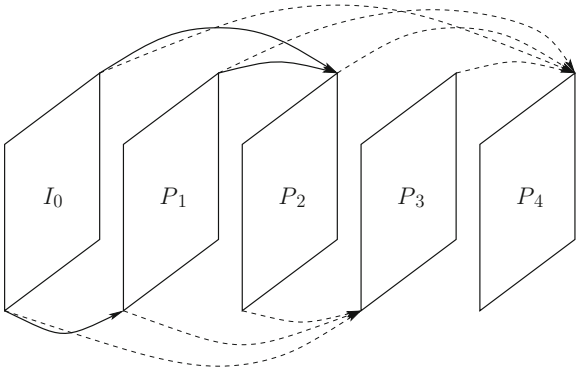

Fig. 12.4 An illustration of multi-reference frames

errors in the transform and inverse transform. It is also slow due to many floating point multiplications. H.264 allows 4×4 blocks and various predictions; even intra coding relies on spatial prediction followed by transform coding. Hence, it is very sensitive to prediction shift. For example, a 4×4 block may be predicted from a neighboring intra block, and the neighboring block may itself be predicted from another neighboring block, and so on. As a result, the prediction shift could be accumulated, causing a large error.

Given the powerful and accurate P- and I- prediction schemes in H.264, it is recognized that the spatial correlation in residual pixels is typically very low. Hence, a simple integer-precision 4×4 DCT is sufficient to compact the energy. The integer arithmetic allows exact inverse transform on all processors and eliminates encoder/decoder mismatch problems in previous transform-based codecs. H.264 also provides a quantization scheme with nonlinear *step-size*s to obtain accurate rate control at both the high and low ends of the quantization scale.

The 4×4 transforms in H.264 approximate the DCT and IDCT. They involve only integer, 16-bit arithmetic operations. They can be implemented very efficiently.

As discussed in Chap. 8, the 2D DCT is separable: it can be realized by two consecutive 1D transforms, i.e., in the vertical direction first and then the horizontal direction. This can be implemented by two matrix multiplications: $\mathbf{F} = \mathbf{T} \times \mathbf{f} \times \mathbf{T}^T$, where $\mathbf{f}$ is the input data, $\mathbf{F}$ is the transformed data, and $\mathbf{T}$ is the so-called DCT-matrix. The DCT-matrix is orthonormal, i.e., all the rows are orthogonal, and they all have norm 1.

The 4×4 DCT-matrix T_4 can be written as:

$$\mathbf{T_4} = \begin{bmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & -c \end{bmatrix}$$

where $a = 1/2$, $b = \sqrt{\frac{1}{2}} \cos \frac{\pi}{8}$, and $c = \sqrt{\frac{1}{2}} \cos \frac{3\pi}{8}$.

To derive a scaled 4×4 integer transform that approximates $\mathbf{T_4}$, we can simply scale the entries of $\mathbf{T_4}$ up and round them to nearest integers [5]:

$$\mathbf{H} = \text{round}(\alpha \cdot \mathbf{T_4}). \tag{12.1}$$

When $\alpha = 26$, we have

$$\mathbf{H} = \begin{bmatrix} 13 & 13 & 13 & 13 \\ 17 & 7 & -7 & -17 \\ 13 & -13 & -13 & 13 \\ 7 & -17 & 17 & -7 \end{bmatrix}$$

Similar to T_4, this matrix has some nice properties: all its rows are orthogonal; they also have the same norm, because $4 \times 13^2 = 2 \times (17^2 + 7^2)$. However, this matrix has a dynamic range gain of 52 (i.e., 4×13). Since it is used twice in $\mathbf{F} = \mathbf{H} \times \mathbf{f} \times \mathbf{H}^T$ in order to transform the columns and then the rows of $\mathbf{f}$, the total gain is $52^2 = 2704$. Because $\log_2 2704 \approx 11.4$, it would require 12 more bits for coefficients in $\mathbf{F}$ than the number of bits required for data in the original $\mathbf{f}$. This would make the 16-bit arithmetic insufficient, and would require 32-bit arithmetic.

Hence, it is proposed in [5] that $\alpha = 2.5$ in Eq. 12.1. This yields

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \tag{12.2}$$

This new matrix $\mathbf{H}$ is still orthogonal, although its rows no longer have the same norm. To restore the orthonormal property, we could simply derive the following matrix $\bar{\mathbf{H}}$ by dividing all row entries in $\mathbf{H}$ by $\sqrt{\sum_j H_{ij}^2}$, where H_{ij}, is the jth entry of the ith row in $\mathbf{H}$. However, this would no longer be an integer transform.

$$\bar{\mathbf{H}} = \begin{bmatrix} 1/2 & 1/2 & 1/2 & 1/2 \\ 2/\sqrt{10} & 1/\sqrt{10} & -1/\sqrt{10} & -2/\sqrt{10} \\ 1/2 & -1/2 & -1/2 & 1/2 \\ 1/\sqrt{10} & -2/\sqrt{10} & 2/\sqrt{10} & -1/\sqrt{10} \end{bmatrix}$$

In the H.264 implementation, this normalization issue is postponed. It is merged into the quantization process in the next step, since we can simply adjust the values in the quantization matrix to achieve both the objectives of quantization and normalization.

Because $\mathbf{H}$ is orthogonal, we could have used $\mathbf{H}^T$ as the inverse transform $\mathbf{H}^{-1}$, as long as the normalization issue is taken care of. Again, since we can also resolve this issue later at the de-quantization step, we simply introduce an ad hoc inverse transform $\mathbf{H_{inv}}$ to use:

$$\mathbf{H_{inv}} = \begin{bmatrix} 1 & 1 & 1 & 1/2 \\ 1 & 1/2 & -1 & -1 \\ 1 & -1/2 & -1 & 1 \\ 1 & -1 & 1 & -1/2 \end{bmatrix} \tag{12.3}$$

$\mathbf{H_{inv}}$ is basically $\mathbf{H}^T$, but with the second and fourth columns scaled down by 1/2. This is because the dynamic range of the input data to $\mathbf{H_{inv}}$ is larger than that of $\mathbf{H}$. Hence, a further scaling down is applied to the columns that would otherwise have a higher dynamic range gain.

H.264 also supports the 8×8 integer transform $\mathbf{H_{8\times 8}}$. It is as in Eq. 12.4. We will use $\mathbf{H}$, the 4×4 version, in our discussions unless otherwise noted.

$$\mathbf{H_{8\times 8}} = \begin{bmatrix} 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 \\ 12 & 10 & 6 & 3 & -3 & -6 & -10 & -12 \\ 8 & 4 & -4 & -8 & -8 & -4 & 4 & 8 \\ 10 & -3 & -12 & -6 & 6 & 12 & 3 & -10 \\ 8 & -8 & -8 & 8 & 8 & -8 & -8 & 8 \\ 6 & -12 & 3 & 10 & -10 & -3 & 12 & -6 \\ 4 & -8 & 8 & -4 & -4 & 8 & -8 & 4 \\ 3 & -6 & 10 & -12 & 12 & -10 & 6 & -3 \end{bmatrix} \tag{12.4}$$

12.1.3 Quantization and Scaling

As in previous video compression standards, *quantization* is used after the transform. Instead of designing simple quantization matrices, H.264 has a more sophisticated design [5] that accomplishes both tasks of the quantization and scaling (normalization) of $\mathbf{H}$.

Integer Transform and Quantization

Let $\mathbf{f}$ be the 4×4 input matrix, and $\hat{\mathbf{F}}$ the transformed and then quantized output. The forward integer transform, scaling and quantization are implemented according to:

$$\hat{\mathbf{F}} = \text{round}\left[(\mathbf{H} \times \mathbf{f} \times \mathbf{H}^T) \cdot \mathbf{M_f}/2^{15}\right]. \tag{12.5}$$

Here, '$\times$' denotes matrix multiplication, while '$\cdot$' denotes element-by-element multiplication. $\mathbf{H}$ is the same as in Eq. 12.2. $\mathbf{M_f}$ is the 4×4 quantization matrix derived from $\mathbf{m}$ which is a 6×3 matrix (see Table 12.1). QP is the *quantizaton parameter*.

For $0 \leq QP < 6$, we have

$$\mathbf{M_f} = \begin{bmatrix} \mathbf{m}(QP,0) & \mathbf{m}(QP,2) & \mathbf{m}(QP,0) & \mathbf{m}(QP,2) \\ \mathbf{m}(QP,2) & \mathbf{m}(QP,1) & \mathbf{m}(QP,2) & \mathbf{m}(QP,1) \\ \mathbf{m}(QP,0) & \mathbf{m}(QP,2) & \mathbf{m}(QP,0) & \mathbf{m}(QP,2) \\ \mathbf{m}(QP,2) & \mathbf{m}(QP,1) & \mathbf{m}(QP,2) & \mathbf{m}(QP,1) \end{bmatrix} \tag{12.6}$$

For $QP \geq 6$, each element $\mathbf{m}(QP,k)$ is replaced by $\mathbf{m}(QP\%6,k)/2^{\lfloor QP/6 \rfloor}$.

Table 12.1 The matrix **m**—used to generate $\mathbf{M_f}$

QP	Positions in $\mathbf{M_f}$ (0, 0), (0, 2) (2, 0), (2, 2)	Positions in $\mathbf{M_f}$ (1, 1), (1, 3) (3, 1), (3, 3)	Remaining $\mathbf{M_f}$ 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

Table 12.2 The matrix **v**—used to generate $\mathbf{V_i}$

QP	Positions in $\mathbf{V_i}$ (0, 0), (0, 2) (2, 0), (2, 2)	Positions in $\mathbf{V_i}$ (1, 1), (1, 3) (3, 1), (3, 3)	Remaining $\mathbf{V_i}$ 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

The quantization is followed by another scaling step, which can be implemented by a right-shift "$\gg 15$".

Inverse Integer Transform and De-quantization

Let $\tilde{\mathbf{f}}$ be the de-quantized and then inversely transformed result. The scaling, de-quantization, and inverse integer transform are implemented according to:

$$\tilde{\mathbf{f}} = \text{round}\left[(\mathbf{H_{inv}} \times (\hat{\mathbf{F}} \cdot \mathbf{V_i}) \times \mathbf{H_{inv}}^T)/2^6\right]. \tag{12.7}$$

$\mathbf{H_{inv}}$ is the same as in Eq. 12.3. $\mathbf{V_i}$ is the 4×4 de-quantization matrix derived from $\mathbf{v}$ which is a 6×3 matrix (see Table 12.2). For $0 \leq QP < 6$, we have

$$\mathbf{V_i} = \begin{bmatrix} \mathbf{v}(QP,0) & \mathbf{v}(QP,2) & \mathbf{v}(QP,0) & \mathbf{v}(QP,2) \\ \mathbf{v}(QP,2) & \mathbf{v}(QP,1) & \mathbf{v}(QP,2) & \mathbf{v}(QP,1) \\ \mathbf{v}(QP,0) & \mathbf{v}(QP,2) & \mathbf{v}(QP,0) & \mathbf{v}(QP,2) \\ \mathbf{v}(QP,2) & \mathbf{v}(QP,1) & \mathbf{v}(QP,2) & \mathbf{v}(QP,1) \end{bmatrix} \tag{12.8}$$

For $QP \geq 6$, each element $\mathbf{v}(QP, k)$ is replaced by $\mathbf{v}(QP\%6, k) \cdot 2^{\lfloor QP/6 \rfloor}$.

The de-quantization is also followed by another scaling step, which can be implemented by a right-shift "$\gg 6$."

12.1.4 Examples of H.264 Integer Transform and Quantization

This section shows some examples of the H.264 integer transform and quantization and their inverse using various quantization parameters QPs. The input data is a 4×4 matrix $\mathbf{f}$ with arbitrary values, the transformed and then quantized coefficients are in $\hat{\mathbf{F}}$. $\mathbf{M_f}$ and $\mathbf{V_i}$ are the quantization and de-quantization matrices, and $\tilde{f}$ is the de-quantized and then inversely transformed output. For comparison, we will also show the compression loss $\boldsymbol{\epsilon} = \mathbf{f} - \tilde{\mathbf{f}}$.

In order to improve the rate-distortion performance, H.264 adopts the dead-zone quantization (also known as *midtread* as discussed in Chap. 8). It can be described as a function that turns a real number x to an integer Z, as follows:

$$Z = \lfloor x + b \rfloor,$$

where x is the scaled value as discussed in the last section. By default, $b = 0.5$, the above function is then equivalent to the *round* function as specified in Eq. (12.5) or (12.7). To minimize the quantization error, H.264 actually adopts adaptive quantization in which the width of the dead-zone can be controlled by b. For example, $b = 1/3$ for intra-coding and $b = 1/6$ for inter-coding. For simplicity, in the following examples, we just use $b = 0.5$.

Figure 12.5a shows the result when $QP = 0$. This is the option that offers the least possible compression loss. The values of $\mathbf{M_f}$ and $\mathbf{V_i}$ are determined according to Eqs. (12.6) and (12.8). Since there is no scale-down of $\mathbf{F}$ values in the quantization step, the reconstructed $\tilde{\mathbf{f}}$ is exactly the same as $\mathbf{f}$, i.e., $\tilde{\mathbf{f}} = \mathbf{f}$.

Figure 12.5b shows the result when $QP = 6$. As expected, compared to corresponding matrix entries for $QP = 0$, the values in $\mathbf{M_f}$ are reduced in half, and the values in $\mathbf{V_i}$ are about twice. The quantization factor is now approximately 1.25 (i.e., equivalent to $qstep \approx 1.25$). As a result, $\tilde{\mathbf{f}} \neq \mathbf{f}$. The slight loss can be observed in $\boldsymbol{\epsilon}$.

Similarly, when $QP = 18$ the result is in Fig. 12.5c. The values in $\mathbf{M_f}$ and the values in $\mathbf{V_i}$ are further decreased or increased, respectively, by a factor of 4 compared to those for $QP = 6$. The quantization factor is approximately 5 (i.e., equivalent to $qstep \approx 5$). The loss shown in $\boldsymbol{\epsilon}$ is more significant now.

Perhaps a more interesting result is shown in Fig. 12.5d when $QP = 30$. The quantization factor is now approximately 20 (i.e., equivalent to $qstep \approx 20$). Except the larger quantized DC value in $\hat{F}$ that remains nonzero, all the AC coefficients become zero. As a result, the reconstructed $\tilde{f}$ has 80 in all entries. The compression loss in $\boldsymbol{\epsilon}$ is no longer acceptable.

12.1.5 Intra Coding

H.264 exploits much more *spatial prediction* than in previous video standards such as MPEG-2 and H.263+. Intra-coded macroblocks are predicted using some of the neighboring reconstructed pixels (using either intra- or inter-coded reconstructed pixels).

(a)

72	82	85	79
74	75	86	82
84	73	78	80
77	81	76	84

$\mathbf{f}$

13107	8066	13107	8066
8066	5243	8066	5243
13107	8066	13107	8066
8066	5243	8066	5243

$\mathbf{M_f}$

507	-12	-2	2
0	-7	-14	5
2	0	-8	-11
-1	8	4	3

$\hat{\mathbf{F}}$

10	13	10	13
13	16	13	16
10	13	10	13
13	16	13	16

$\mathbf{V_i}$

72	82	85	79
74	75	86	82
84	73	78	80
77	81	76	84

$\tilde{\mathbf{f}}$

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

$\epsilon = \mathbf{f} - \tilde{\mathbf{f}}$

(b)

72	82	85	79
74	75	86	82
84	73	78	80
77	81	76	84

$\mathbf{f}$

6554	4033	6554	4033
4033	2622	4033	2622
6554	4033	6554	4033
4033	2622	4033	2622

$\mathbf{M_f}$

254	-6	-1	1
0	-4	-7	3
1	0	-4	-6
0	4	2	1

$\hat{\mathbf{F}}$

20	26	20	26
26	32	26	32
20	26	20	26
26	32	26	32

$\mathbf{V_i}$

72	82	85	79
74	75	86	82
84	74	78	80
77	82	76	84

$\tilde{\mathbf{f}}$

0	0	0	0
0	0	0	0
0	-1	0	0
0	-1	0	0

$\epsilon = \mathbf{f} - \tilde{\mathbf{f}}$

(c)

72	82	85	79
74	75	86	82
84	73	78	80
77	81	76	84

$\mathbf{f}$

1638	1008	1638	1008
1008	655	1008	655
1638	1008	1638	1008
1008	655	1008	655

$\mathbf{M_f}$

63	-2	0	0
0	-1	-2	1
0	0	-1	-1
0	1	0	0

$\hat{\mathbf{F}}$

80	104	80	104
104	128	104	128
80	104	80	104
104	128	104	128

$\mathbf{V_i}$

70	81	86	78
73	73	85	83
82	75	77	82
77	79	74	85

$\tilde{\mathbf{f}}$

2	1	-1	1
1	2	1	-1
2	-2	1	-2
0	2	2	-1

$\epsilon = \mathbf{f} - \tilde{\mathbf{f}}$

(d)

72	82	85	79
74	75	86	82
84	73	78	80
77	81	76	84

$\mathbf{f}$

410	252	410	252
252	164	252	164
410	252	410	252
252	164	252	164

$\mathbf{M_f}$

16	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

$\hat{\mathbf{F}}$

320	416	320	416
416	512	416	512
320	416	320	416
416	512	416	512

$\mathbf{V_i}$

80	80	80	80
80	80	80	80
80	80	80	80
80	80	80	80

$\tilde{\mathbf{f}}$

-8	2	5	-1
-6	-5	6	2
4	-7	-2	0
-3	1	-4	4

$\epsilon = \mathbf{f} - \tilde{\mathbf{f}}$

Fig. 12.5 Examples of H.264 integer transform and quantization with various QPs: **a** QP=0; **b** QP=6; **c** QP=18; **d** QP=30

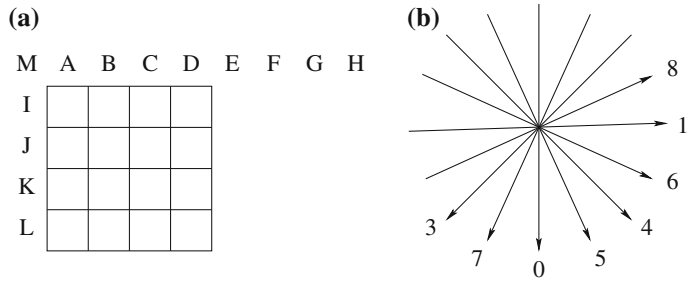

Fig. 12.6 H.264 intra frame prediction. **a** Intra_4 × 4 prediction using neighboring samples A to M. **b** Eight directions for intra_4 × 4 predictions

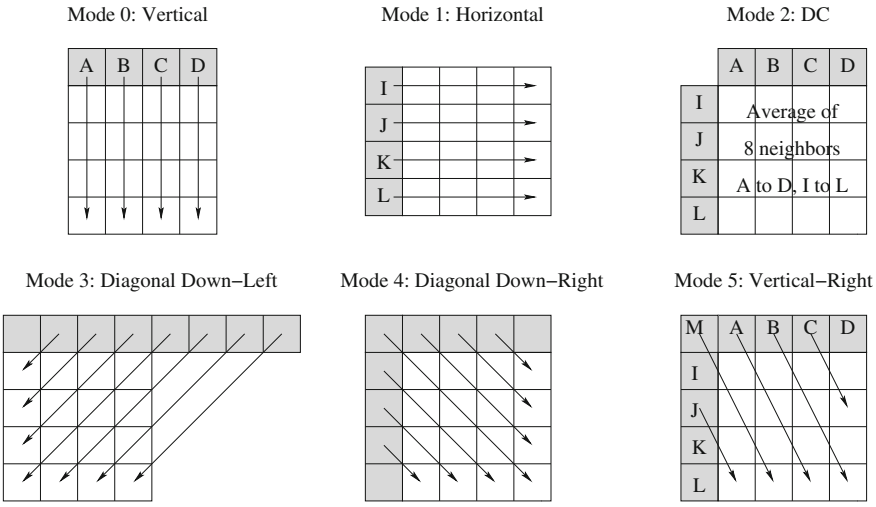

Fig. 12.7 The first six of nine intra_4 × 4 prediction modes in H.264 intra frame prediction

Similar to the variable block sizes in inter-picture motion compensation, different intra prediction block sizes (4 × 4 or 16 × 16) can be chosen for each intra-coded macroblock. As shown in Fig. 12.6, there are nine prediction modes for the 4 × 4 blocks in Intra_4 × 4. Figure 12.7 further illustrates six of them.

Prediction modes 0, 1, 3, and 4 are fairly straightforward. For example, in Mode 0, the value of pixel A will be used as the predicted value for all pixels in the first column (i.e., the column below A), the value of pixel B will be used as the predicted value for all pixels in the second column, etc. Mode 2 (DC) is a special mode in which the average of the eight previously coded neighbors (A to D and I to L) is used as the predicted value for all pixels in the 4 × 4 block.

Mode 5 (Vertical-Right), Mode 6 (Horizontal-Down), Mode 7 (Vertical-Left), and Mode 8 (Horizontal-Up) are similar. As shown in Fig. 12.7, the direction of the prediction in Mode 5 is down and to the right at the ratio of 2:1 (i.e., 2 pixels down and 1 pixel to the right, or approximately 26.6 degrees to the right). This works well

for the pixels at the second and fourth rows. For example, if the row and column indices of the 4×4 block are in the range 0..3, then the prediction value for pixels at [1, 1] and [3, 2] will be the value of A. However, the pixels at the first and third rows will not be able to use a single value from any of the previously coded neighbors. Instead, it must be extrapolated from two of them. For example, the prediction value for pixels at [0, 0] and [2, 1] will be a proportional combination of the values of M and A.

For each prediction mode, the predicted value and the actual value will be compared to produce the prediction error. The mode that produces the least prediction error will be chosen as the prediction mode for the block. The prediction errors (residuals) are then sent to transform coding where the 4×4 integer transform is employed. Each 4×4 block in a macroblock may have a different prediction mode. The sophisticated intra-prediction is powerful as it drastically reduces the amount of data to be transmitted when temporal prediction fails.

There are only four prediction modes for the 16×16 blocks in Intra_16×16. Mode 0 (Vertical), Mode 1 (Horizontal) and Mode 2 (DC) are very similar to Intra_4×4 above except the larger block size. For Mode 3 (Plane) which is unique to 16×16 blocks, a plane (linear) function is fitted to the upper and left samples in the 16×16 block as the prediction.

In summary, the following four modes are specified for intra coding:

- Intra_4×4 for luma macroblocks
- Intra_16×16 for luma macroblocks
- Intra coding for chroma macroblocks—it uses the same four prediction modes as in Intra_16×16 luma. The prediction block size is 8×8 for 4:2:0, 8×16 for 4:2:2, and 16×16 for 4:4:4 chroma sampling.
- I_PCM (Pulse Code Modulation)—bypass the spatial prediction and transform coding, and directly send the PCM coded (fixed-length) luma and chroma pixel values. It is invoked in rare cases when other prediction modes failed to produce any data compression/reduction.

12.1.6 In-Loop Deblocking Filtering

One of the prominent deficiencies of the block-based coding methods is the generation of unwanted visible block structures. Pixels at the block boundaries are usually reconstructed less accurately: they tend to look like the interior pixels in the same block, hence the artificial appearance of blocks.

H.264 specifies a signal-adaptive deblocking filter in which a set of filters is applied on the 4×4 block edges. Filter length, strength, and type (deblocking/smoothing) vary, depending on the macroblock coding parameters (intra- or inter-coded, reference-frame differences, coefficients coded, etc.) and spatial activity (edge detection), so that blocking artifacts are eliminated without distorting visual features. The H.264 deblocking filter is important in increasing the subjective quality of the videos.

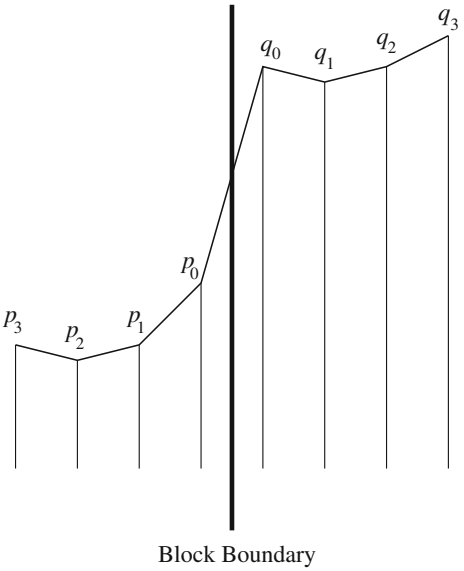

Fig. 12.8 Deblocking of a 1-D edge on the block boundary

As shown in Fig. 12.1, in H.264 the deblocking filtering takes place in the loop, after the inverse transform in the encoder, before the decoded block data are fed to Motion Estimation.

Figure 12.8 depicts a simplified 1-D edge, where the height of the pixels p_0, q_0, etc., indicates their value. The function of *deblocking filtering* is basically smoothing of the block edges. For example, a "four-tap filtering" will take some weighted average of the values of p_1, p_0, q_0, and q_1 to generate new p_0 or q_0.

Apparently, real edges across the block boundary will need to be protected from the deblocking filtering. The deblocking filtering on p_0 and q_0 will be applied only if all the following criteria are met:

$$
\begin{aligned}
|p_0 - q_0| &< \alpha(QP), \\
|p_0 - p_1| &< \beta(QP), \\
|q_0 - q_1| &< \beta(QP),
\end{aligned}
$$

where α and β are thresholds, and they are functions of the quantization parameter QP as defined in the standard. They are lower when QP is smaller. This is because when QP is small, a relatively significant difference, e.g., $|p_0 - q_0|$ is likely caused by a real edge.

In addition to p_0 and q_0, the deblocking filtering on p_1 or q_1 will be applied if

$$
|p_0 - p_2| < \beta(QP) \quad \text{or} \quad |q_0 - q_2| < \beta(QP)
$$

Table 12.3 The 0th order Exp-Golomb codewords (EG_0)

Unsigned N	Signed N	Codeword
0	0	1
1	1	010
2	−1	011
3	2	00100
4	−2	00101
5	3	00110
6	−3	00111
7	4	0001000
8	−4	0001001
…	…	…

12.1.7 Entropy Coding

H.264 has developed a set of sophisticated entropy coding methods. When entropy_coding_mode = 0, a simple Exponential-Golomb (Exp-Golomb) code is used for header data, motion vectors, and other nonresidual data, while the more complex *Context-Adaptive Variable Length Coding (CAVLC)* is used for quantized residual coefficients. When entropy_coding_mode = 1, *Context-Adaptive Binary Arithmetic Coding (CABAC)* is used (see Sect. 12.1.9).

Simple Exp-Golomb code

The simple Exponential-Golomb (Exp-Golomb) code that is used for header data, etc., is the so-called 0th order Exp-Golomb code (EG_0). It is a binary code, and it consists of three parts:

[Prefix] [1] [Suffix]

The prefix is a sequence of l zeros. Given an unsigned (positive) number N to be coded, $l = \lfloor \log_2(N+1) \rfloor$. The Suffix S is the binary number $N+1-2^l$ represented in l bits.

As shown in Table 12.3, if an unsigned $N = 4$, then $l = \lfloor \log_2(4+1) \rfloor = 2$, the Prefix is 00; the Suffix S is the binary number $S = 4+1-2^2 = 1$ represented in 2 bits, i.e., 01. Hence, the Exp-Golomb code for $N = 4$ is 00101.

To decode the Exp-Golomb codeword EG_0 for unsigned N, the following steps can be followed:

1. Read in the sequence of consecutive zeros, l = number_of_zeros.
2. Skip the next '1'.
3. Read in the next l bits and assign to S.
4. $N = S - 1 + 2^l$.

The unsigned numbers are used to indicate, e.g., macroblock type, reference frame index, etc. For signed numbers, e.g., motion vector difference, they will simply be

Table 12.4 First- and second-order Exp-Golomb codewords (EG_1 and EG_2)

Unsigned N	EG_1 Codeword	EG_2 Codeword
0	10	100
1	11	101
2	0100	110
3	0101	111
4	0110	01000
5	0111	01001
6	001000	01010
7	001001	01011
8	001010	01100
9	001011	01101
10	001100	01110
11	001101	01111
12	001110	0010000
13	001111	0010001
14	00010000	0010010
15	00010001	0010011
...	...	...

squeezed in to produce a new set of table entries as listed in the second column (Table 12.3).

*k*th Order Exp-Golomb Code

In general, the Exp-Golomb code can have a higher order, i.e., the kth order EG_k. Similarly, it is a binary code and consists of three parts: [Prefix] [1] [Suffix]. The Prefix is a sequence of l zeros. Given an unsigned (positive) number N to be coded, $l = \lfloor \log_2(N/2^k + 1) \rfloor$. The Suffix S is the binary number $N + 2^k(1 - 2^l)$ represented in $l + k$ bits.

For example, the EG_1 code for $N = 4$ is 0110. It is because $l = \lfloor \log_2(4/2^1 + 1) \rfloor$ = 1, the prefix is 0; the suffix is the binary representation of $4 + 2^1(1 - 2^1) = 2$, in $l + k = 1 + 1 = 2$ bits it is 10. Table 12.4 provides some examples of the first- and second-order Exp-Golomb codes for non-negative numbers.

To decode the kth order Exp-Golomb codeword EG_k for unsigned N, the following steps can be followed:

1. Read in the sequence of consecutive zeros, l = number_of_zeros.
2. Skip the next '1'.
3. Read in the next $l + k$ bits and assign to S.
4. $N = S - 2^k(1 - 2^l)$.

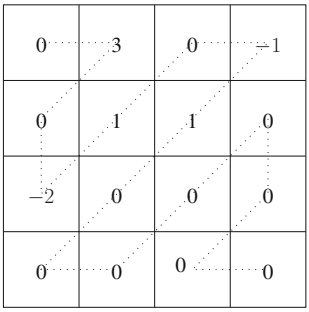

Fig. 12.9 Example: A 4 × 4 block of data for CAVLC encoder

12.1.8 Context-Adaptive Variable Length Coding (CAVLC)

Previous video coding standards such as MPEG-2 and H.263 use fixed VLC. In CAVLC [6,7], multiple VLC tables are predefined for each data type (zero-runs, levels, etc.), and predefined rules predict the optimal VLC table based on the context, e.g., previously decoded neighboring blocks.

It is known that the matrices (by default 4×4) that contain the quantized frequency coefficients of the residual data are typically sparse, i.e., contain many zeros. Even when they are nonzero, the quantized coefficients for higher frequencies are often $+1$ or -1 (the so-called "trailing_1s"). CAVLC exploits these characteristics by carefully extracting the following parameters from the current block data:

- Total number of nonzero coefficients (TotalCoeff) and number of trailing ± 1s (Trailing_1s).
- Signs of the Trailing_1s.
- Level (sign and magnitude) of the other nonzero coefficients (not Trailing_1s).
- Total number of zeros before the last nonzero coefficient.
- Run of zeros (zeros_left and run_before) before each nonzero coefficient.

Figure 12.9 shows an example of a 4×4 block of the residual data after transform and quantization. After the zigzag scan, the 1-D sequence is: 0 3 0 −2 1 0 −1 1 0 0 0 0 0 0 0 0. Table 12.5 gives details of the CAVLC code generated. The nonzero coefficients are processed in reverse order, i.e., the last one indexed by '4' (Trailing_1[4]) is examined first, and so forth.

We will briefly explain the code generation process for the above example.

- A code 0000100 is generated for TotalCoeffs = 5, Trailing_1s = 3. These are looked up from "Table 9–5—coeff_token mapping to TotalCoeff and TrailingOnes" in the H.264 standard (2003). It is observed that, in general, the numbers of nonzero coefficients in the neighboring blocks are similar, hence the code for coeff_token is context-adaptive. For each pair of TotalCoeffs and Trailing_1s, its code can be assigned one of the four possible values depending on the numbers of nonzero coefficients in the blocks above and to the left of the current block. If the neighboring blocks have a small number of nonzeros, then a code assignment favoring

Table 12.5 CAVLC code generation for data from Fig. 12.9

Data	Value	Code
coeff_token	TotalCoeffs = 5, Trailing_1s = 3	0000100
Trailing_1 [4] Sign	+	0
Trailing_1 [3] Sign	−	1
Trailing_1 [2] Sign	+	0
Level [1]	−2 (SuffixLength = 0)	0001 (prefix)
Level [0]	3 (SuffixLength = 1)	001 (prefix) 0 (suffix)
Total zeros	3	111
run_before [4]	zeros_left = 3, run_before = 0	11
run_before [3]	zeros_left = 3, run_before = 1	10
run_before [2]	zeros_left = 2, run_before = 0	1
run_before [1]	zeros_left = 2, run_before = 1	01
run_before [0]	zeros_left = 1, run_before = 1	No code required

the small TotalCoeffs in the current block will be used (i.e., small TotalCoeffs are assigned very short codes and large TotalCoeffs are assigned particularly long codes), and vice versa. In this example, it is assumed that the number of nonzero coefficients in the two neighboring blocks is less than 2.

- Sign '+' is assigned the code 0, and '−' the code 1.
- The choice of the VLC code for Level is again context-adaptive, it depends on the magnitudes of the recently coded Levels. In reverse order, the first nonzero coefficient is −2. Initially, SuffixLength = 0, so the code for −2 is 0001 (prefix). Afterwards, SuffixLength is increased by 1, hence the next nonzero 3 gets the code 001 (prefix) 0 (suffix). The magnitude of levels tends to increase (when examined in reverse order), so the SuffixLength is increased adaptively to accommodate larger magnitudes. For further details, the readers are referred to [4,6,7].
- The total number of zeros is 3. It gets the code 111.
- The last five rows in Table 12.5 record the information for the runs of zeros in the current block. For example, for the last nonzero coefficient '1', 3 zeros are in front of it, and no zero is immediately in front of it. The code 11 is looked up from Tables 9–10 in the H.264 standard. To illustrate, we extracted part of it below as Table 12.6. This should also explain the codes in the next three rows, 10, 1, and 01. Come down to the last row, only one zero is left for the only (last) nonzero coefficient, the encoder and decoder can unambiguously determine it, so no code for run_before is needed.

For this example, the resulting sequence of the code is 0000100 0 1 0 0001 001 0 111 11 10 1 01. Based on it, the decoder is able to reproduce the block data.

Table 12.6 Code for various run_before

run_before	zeros_left						
	1	2	3	4	5	6	> 6
0	1	1	11	11	11	11	111
1	0	01	10	10	10	000	110
2	—	00	01	01	011	001	101
3	—	—	00	001	010	011	100
4	—	—	—	000	001	010	011
5	—	—	—	—	000	101	010
…							

12.1.9 Context-Adaptive Binary Arithmetic Coding (CABAC)

The VLC-based entropy coding methods (including CAVLC) are inefficient in dealing with symbols with a probability greater than 0.5, because usually a minimum of 1 bit has to be assigned to each symbol, which could be much larger than its *self information* measured by $\log_2 \frac{1}{p_i}$, where p_i is the symbol's probability. For better coding efficiency in H.264 Main and High profiles, Context-Adaptive Binary Arithmetic Coding (CABAC) [8] is used for some data and quantized residual coefficients when entropy_coding_mode = 1.

As shown in Fig. 12.10, CABAC has three major components:

- **Binarization.** All nonbinary data are converted to binary bit (*bin*) strings first since CABAC uses Binary Arithmetic Coding. Five schemes can be used for the binarization: (a) Unary (U)—for $N \geq 0$, it is N 1s followed by a terminating 0. For example, it is 111110 for 5. (b) Truncated Unary (TU)—similar to U, but without the terminating 0. (c) kth order Exp-Golomb code. (d) The concatenation of (a) and (c). (e) Fixed-Length binary scheme.
- **Context Modeling.** This step deals with the context model selection and access. The *Regular Coding Mode* is for most symbols, e.g., macroblock type, mvd, information about prediction modes, information about slice and macroblock control, and residual data. Various "context models" are built to store the conditional probabilities for the bins of the binarized symbol to be 1 or 0. The probability models are derived from the statistics of the context, i.e., recently coded symbols and bins. The *Bypass coding mode* uses no context model; it is used to speed up the coding process.
- **Binary Arithmetic Coding.** For efficiency, a binary arithmetic coding method is developed [8]. Below is a brief description of *Binary Arithmetic Coding* in H.264. As shown in Chap. 7, arithmetic coding involves recursive subdivisions of the current range. The numerous multiplications involved there is its main disadvantage in terms of the computational cost when compared with other entropy coding methods. Binary arithmetic coding only involves two symbols, so the number of multiplications is greatly reduced.

  In the binary case, we name the two symbols *LPS (Least Probable Symbol)* and *MPS (Most Probable Symbol)*, the range as R, the lower bound of R as L. (Note:

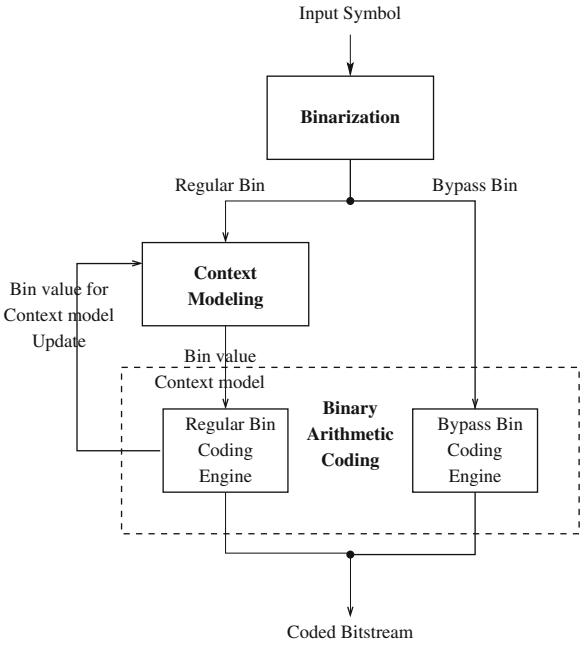

Fig. 12.10 Block diagram of CABAC in H.264

LPS is the upper interval and MPS is the lower interval in R.) If the probability of the LPS is P_{LPS}, then $P_{MPS} = 1 - P_{LPS}$, and the following procedure can be employed for the generation of the next range given the new symbol S:

PROCEDURE Calculating Ranges in Binary Arithmetic Coding

BEGIN

If S is MPS

$$R = R \times (1 - P_{LPS});$$

Else // S is LPS.

$$L = L + R \times (1 - P_{LPS});$$

$$R = R \times P_{LPS};$$

END

However, the multiplication in $R \times P_{LPS}$ would be computational expensive. Various "multiplication-free" binary arithmetic coding schemes have been developed, for example, the Q-coder for binary images, its improved QM-coder, and the MQ-coder adopted in JPEG2000. The H.264 binary arithmetic coding

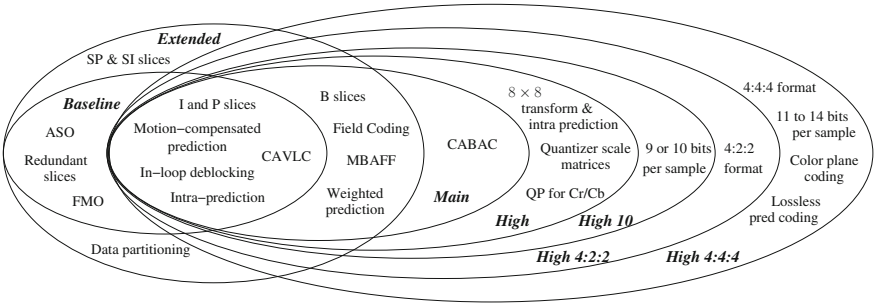

Fig. 12.11 H.264 profiles

method developed by Marpe et al. [8] is the so-called *M-coder (Modulo Coder)*. Here, the multiplication in $R \times P_{LPS}$ is replaced by a table look-up. In the Regular Bin Coding mode, the table has 4×64 pre-calculated product values to allow four different values for R and 64 values for p_{LPS}. (The Bypass Bin Coding mode assumes a uniform probability model, i.e., $P_{MPS} \approx P_{LPS}$ in order to simplify and speed up the process.)

Apparently, due to the limited size of the look-up table, the precision of the product values is limited. These multiplication-free methods are hence known as performing *Reduced-precision Arithmetic Coding*. Previous study shows that the impact of the reduced-precision on the code length is minimal.

The implementation of CABAC has a tremendous amount of details. Readers are referred to [8] for detailed discussions.

12.1.10 H.264 Profiles

As before, a number of profiles are provided to suit the needs of various applications ranging from mobile devices to broadcast HDTV. Figure 12.11 provides an overview of the H.264 profiles [4,9].

Baseline Profile

The Baseline profile of H.264 is intended for real-time conversational applications, such as videoconferencing. It contains all the core coding tools of H.264 discussed above and the following additional error-resilience tools, to allow for error-prone carriers such as IP and wireless networks:

- **Arbitrary slice order (ASO).** The decoding order of slices within a picture may not follow monotonic increasing order. This allows decoding of out-of-order packets in a packet-switched network, thus reducing latency.
- **Flexible macroblock order (FMO).** Macroblocks can be decoded in any order, such as checkerboard patterns, not just raster scan order. This is useful on error-prone networks, so that loss of a slice results in loss of macroblocks scattered

in the picture, which can easily be masked from human eyes. This feature can also help reduce jitter and latency, as the decoder may decide not to wait for late slices and still be able to produce acceptable pictures.
- **Redundant slices.** Redundant copies of the slices can be decoded, to further improve error resilience.

Main Profile

The main profile defined by H.264 represents non-low-delay applications such as standard definition (SD) digital broadcast TV and stored-medium. The main profile contains all Baseline profile features (except ASO, FMO, and redundant slices) plus the following non-low-delay and higher complexity features, for maximum compression efficiency:
- **B slices.** The bi-prediction mode in H.264 has been made more flexible than in existing standards. Bi-predicted pictures can also be used as reference frames. Two reference frames for each macroblock can be in any temporal direction, as long as they are available in the reference frame buffer. Hence, in addition to the normal forward + backward bi-prediction, it is legal to have backward + backward or forward + forward prediction as well.
- **Context-Adaptive Binary Arithmetic Coding (CABAC).** This coding mode replaces VLC-based entropy coding with binary arithmetic coding that uses a different adaptive statistics model for different data types and contexts.
- **Weighted Prediction.** Global weights (multiplier and an offset) for modifying the motion-compensated prediction samples can be specified for each slice, to predict lighting changes and other global effects, such as fading.

Extended Profile

The eXtended profile (or profile X) is designed for the new video streaming applications. This profile allows non-low-delay features, bitstream switching features, and also more error-resilience tools. It includes all Baseline profile features plus the following:
- B slices.
- Weighted prediction.
- **Slice data partitioning.** This partitions slice data with different importance into separate sequences (header information, residual information) so that more important data can be transmitted on more reliable channels.
- **SP (Switching P) and SI (Switching I) slice types.** These are slices that contain special temporal prediction modes, to allow efficient switching of bitstreams produced by different decoders. They also facilitate fast forward/backward, and random access.

High Profiles

H.264/AVC also has four high profiles for applications that demand higher video qualities, i.e., High Definition (HD).

- **High Profile** This profile is adopted by the Blu-ray Disc format and DVB HDTV broadcast. It supports 8×8 integer transform for the parts of the pictures that do not have much detail, and 4×4 integer transform for the parts that do have details. It also allows 8×8 Intra prediction for better coding efficiency especially for higher resolution videos. It provides adjustable quantizer scale matrices, and separate quantizer parameters for Cb and Cr. It also supports monochrome video (4:0:0).
- **High 10 Profile** Supports 9 or 10 bits per sample.
- **High 4:2:2 Profile** Supports 4:2:2 chroma subsampling.
- **High 4:4:4 Predictive Profile** It supports up to 4:4:4 chroma sampling, up to 14 bits per sample, coding of separate color planes, and efficient lossless predictive coding.

12.1.11 H.264 Scalable Video Coding

The *Scalable Video Coding (SVC)* extension of the H.264/AVC standard was approved in 2007 [10]. It provides the bitstream scalability which is especially important for multimedia data transmission through various networks that may have very different bandwidths.

Similar to MPEG-2 and MPEG-4, H.264/AVC SVC provides *temporal scalability*, *spatial scalability*, *quality scalability*, and their possible combinations. Compared to previous standards, the coding efficiency is greatly improved. Other functions such as bit rate and power adaptation, and graceful degradation in lossy network transmissions are also provided.

We covered the issues of temporal scalability, spatial scalability, quality (SNR) scalability, and their possible combinations in sufficient details under MPEG-2 in Chap. 11. Since the fundamental concepts and approaches are very similar, we will not discuss this topic in detail in this chapter. For more information about H264/AVC SVC, readers are referred to [10] and Annex G extension of the H.264/AVC standard.

12.1.12 H.264 Multiview Video Coding

Multiview Video Coding (MVC) is an emerging issue. It has potential applications in some new areas such as *Free Viewpoint Video (FVV)* where users can specify their preferred views. Merkle et al. [11] described some possible MVC prediction structures. Figure 12.12 shows a small example in which there are only four views. The two most important features are:

- **Interview Prediction** Since there is apparent redundancy among the multiple views, the IPPP structure, for example, can be employed for the so-called Key

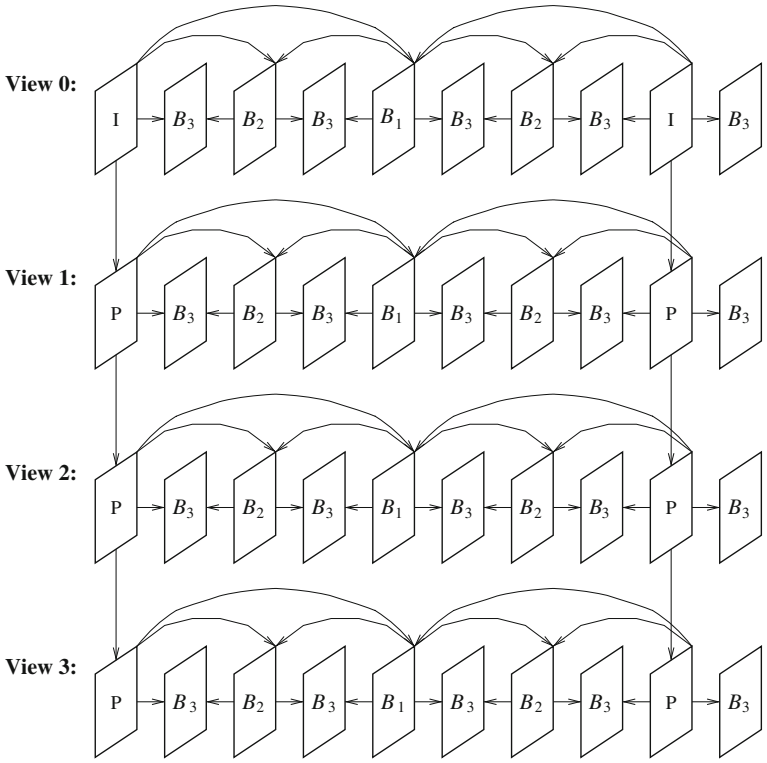

Fig. 12.12 H.264 MVC prediction structure

Pictures (the first and ninth pictures in each view in the figure). This Interview Prediction structure can of course be extended to other structures, e.g.,IBBP. This would be even more beneficial when many more views are involved.

- **Hierarchical B Pictures** For temporal prediction in each view, a hierarchy of B picture, e.g., B_1, B_2, B_3, similar to the ones discussed in Sect. 12.1.1 can be employed. It is made feasible because H.264/AVC is more flexible in supporting various prediction schemes at picture/sequence level. As discussed earlier, increasingly larger quantization parameters can usually be applied down the hierarchy to control the compression efficiency.

12.2 H.265

HEVC (High Efficiency Video Coding) [12, 13] was the latest standard jointly developed by the Joint Collaborative Team on Video Coding (JCT-VC) from the groups of ITU-T VCEG (Video Coding Experts Group) and ISO/IEC MPEG. The final draft of the standard was produced in January 2013. In ISO/IEC, HEVC became MPEG-H

Part 2 (ISO/IEC 23008-2). It is also known as ITU-T Recommendation H.265 [14], which is the term we will use in this book.

The development of this new standard was largely motivated by two factors: (a) The need to further improve coding efficiency due to ever increasing video resolution (e.g., up to $8k \times 4k$ in UHDTV). (b) The need to speed up the more complex coding/decoding methods by exploiting the increasingly available parallel processing devices and algorithms. The initial goal was a further 50 % reduction of the size of the compressed video (with the same visual quality) from H.264, and it was reported that this goal was exceeded. With their superior compression performance over MPEG-2, H.264 and H.265 are currently the leading candidates to carry a whole range of video contents on many potential applications.

At this point, the default format for color video in H.265 is YCbCr. In main profiles, chroma subsampling is 4:2:0.

Main features of H.265 are:

- Variable block-size motion compensation, from 4×4 up to 64×64 in luma images. The macroblock structure is replaced by a quadtree structure of coding blocks at various levels and sizes.
- Exploration of parallel processing.
- Integer transform in various sizes, from 4×4, 8×8, 16×16 to 32×32.
- Improved interpolation methods for the quarter-pixel accuracy in motion vectors.
- Expanded directional spatial prediction (33 angular directions) for intra coding.
- The potential use of DST (Discrete Sine Transform) in luma intra coding.
- In-loop filters including deblocking-filtering and SAO (Sample Adaptive Offset).
- Only CABAC (Context-Adaptive Binary Arithmetic Coding) will be used, i.e., no more CAVLC.

12.2.1 Motion Compensation

As in previous video coding standards, H.265 still uses the technology of *hybrid coding*, i.e., a combination of inter/intra predictions and 2D transform coding on residual errors.

Variable block sizes are used in inter/intra predictions as in H.264. However, more partitions of the prediction and transform blocks are encouraged in order to reduce the prediction errors. Unlike previous video coding standards, H.265 does not use the simple and fixed structure of **macroblocks**. Instead, a quadtree hierarchy of various blocks is introduced as below for its efficiency.

- **CTB** and **CTU** (Coding Tree Block and Coding Tree Unit): CTB is the largest block, the root in the quadtree hierarchy. The size of the luma CTB is $N \times N$, where N can be 16, 32, or 64. The chroma CTB is half-size, i.e., $N/2 \times N/2$. A CTU consists of 1 luma CTB and 2 chroma CTBs.
- **CB** and **CU** (Coding Block and Coding Unit): The CTB consists of CBs organized in the quadtree structure. CB is a square block that can be as small as 8×8 in luma and 4×4 in chroma images. The CBs in a CTB are traversed and coded in Z-order. One luma CB and two chroma CBs form a CU.

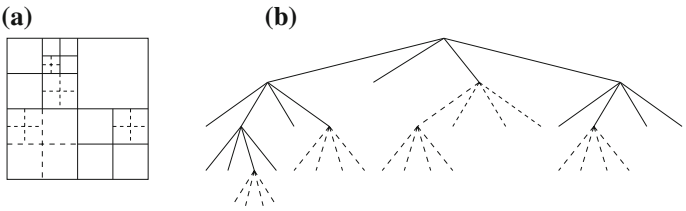

Fig. 12.13 Partitioning of a CTB. **a** The CTB and its partitioning (*solid lines* CB boundaries, *dotted line* TB boundaries). **b** The corresponding quadtree

- **PB** and **PU** (Prediction Block and Prediction Unit): A CB can be further split into PBs for the purpose of prediction. The prediction mode for a CU can be intra-picture (spatial) or inter-picture (temporal). For the intra prediction, the sizes of CB and PB are usually the same; except when the CB is 8×8, a split into four PBs is allowed so that each PB may have a different prediction mode. For the inter prediction, a luma or chroma CB can be split into one, two, or four PBs, i.e., the PBs may not be square albeit always rectangular. The PU contains the luma and chroma PBs and their prediction syntax.
- **TB** and **TU** (Transform Block and Transform Unit): A CB can be further split into TBs for the purpose of transform coding of residual errors. This is represented in the same quadtree structure, hence it is very efficient. The range of the TB size is 32×32 down to 4×4. In H.265, the TBs are allowed to span across PB boundaries in inter-predicted CUs in order to gain higher coding efficiency. A TU consists of TBs from luma and chroma images.

Figure 12.13 illustrates an example in which a CTB is partitioned into CBs and then further into TBs in a quadtree structure. In this example, the original CTB is 64×64 and the smallest TB is 4×4.

Slices and Tiles

As in H.264, H.265 supports Slices of any length consisting of a sequence of CTUs (Fig. 12.14a). They can be I-slices, P-slices, or B-slices.

In addition to the slices, the concept of *Tile* is introduced to facilitate parallel processing amongst multiple Tiles. A tile is a rectangular structure consisting of CTUs (Fig. 12.14b); it may also contain multiple slices.

An additional feature is the inclusion of the *Wavefront Parallel Processing (WPP)* technology. Basically, rows of CTUs can be processed in parallel, in multiple threads in the wavefront manner shown in (Fig. 12.14c).

For the time being, the standard does not allow the mixed use of both Tiles and Wavefronts.

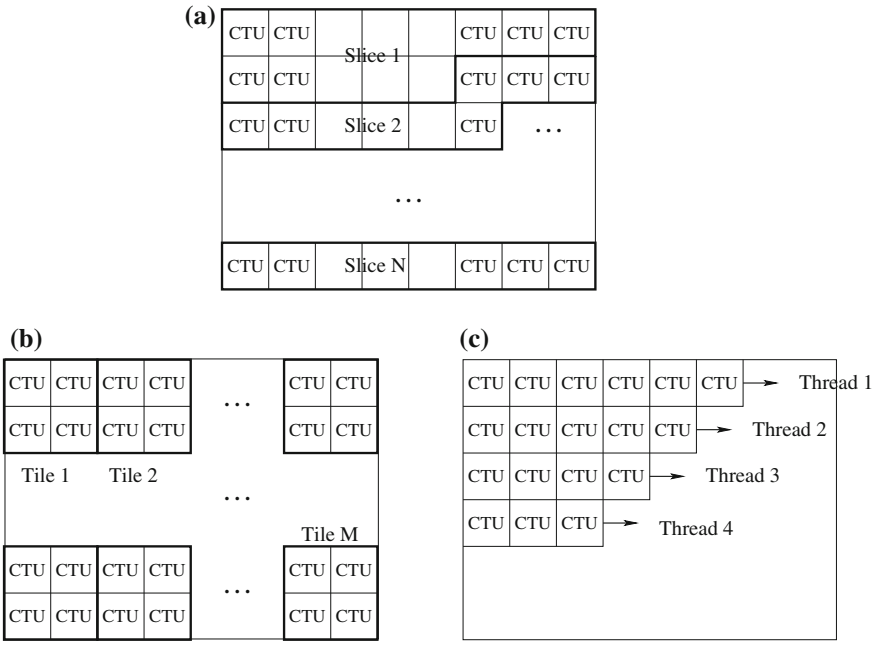

Fig. 12.14 Slices, tiles, and wavefront parallel processing (WPP) in H.265. **a** Slices, **b** Tiles, **c** Wavefronts

Quarter-Pixel Precision in Luma Images

In inter-picture prediction for luma images, the precision for motion vectors is again at quarter-pixel as in H.264. The values at subpixel positions are derived through interpolations. As shown in Table 12.7, an eight-tap filter **hfilter** is used for half-pixel positions (e.g., position b in Fig. 12.15), and a seven-tap filter **qfilter** is used for quarter-pixel positions (e.g., positions a and c in Fig. 12.15).

As shown below, all values at subpixel positions are derived through separable filtering steps vertically and horizontally. This is different from H.264 which uses six-tap filtering to get the values at half-pixel positions, and then averaging to obtain the values at quarter-pixel positions.

The values at positions a, b, and c can be derived using the following:

$$a_{i,j} = \sum_{t=-3}^{3} A_{i,j+t} \cdot \text{qfilter}[t], \tag{12.9}$$

$$b_{i,j} = \sum_{t=-3}^{4} A_{i,j+t} \cdot \text{hfilter}[t], \tag{12.10}$$

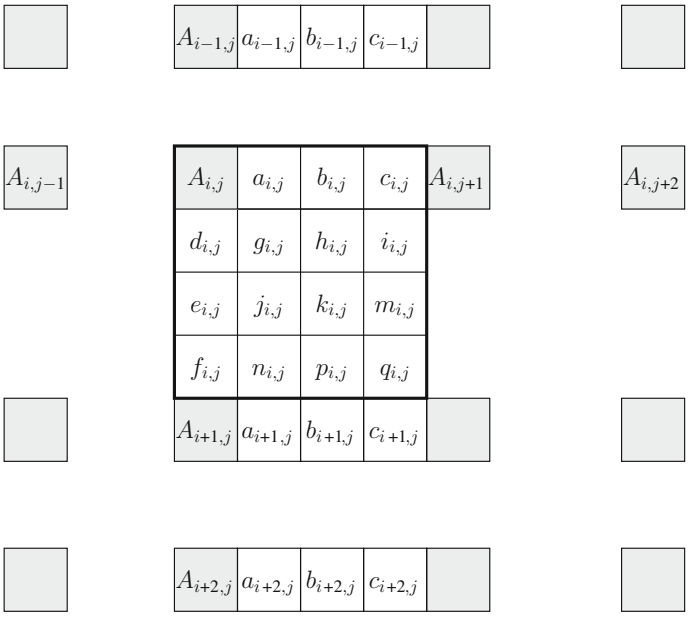

Fig. 12.15 Interpolation for fractional samples in H.265. *Lower-case letters* ($a_{i,j}$, $b_{i,j}$, etc.) indicate pixels at quarter-pixel and half-pixel positions. $A_{i,j}$, $A_{i,j+1}$, etc. indicate pixels on the image grid. (To save space, $A_{i,j-2}$, $A_{i,j+3}$, etc., are not drawn in the figure, although they will be used in the calculation)

Table 12.7 Filters for sample interpolations in Luma images in H.265

Filter	Number of taps	Array index							
		−3	−2	−1	0	1	2	3	4
hfilter	8	−1	4	−11	40	40	−11	4	−1
qfilter	7	−1	4	−10	58	17	−5	1	

$$c_{i,j} = \sum_{t=-2}^{4} A_{i,j+t} \cdot \text{qfilter}[1-t]. \tag{12.11}$$

Numerically,

$$\begin{aligned}
a_{i,j} &= -A_{i,j-3} + 4 \cdot A_{i,j-2} - 10 \cdot A_{i,j-1} + 58 \cdot A_{i,j} + 17 \cdot A_{i,j+1} - 5 \cdot A_{i,j+2} + A_{i,j+3}, \\
b_{i,j} &= -A_{i,j-3} + 4 \cdot A_{i,j-2} - 11 \cdot A_{i,j-1} + 40 \cdot A_{i,j} + 40 \cdot A_{i,j+1} - 11 \cdot A_{i,j+2} \\
&\quad + 4 \cdot A_{i,j+3} - A_{i,j+4}, \\
c_{i,j} &= A_{i,j-2} - 5 \cdot A_{i,j-1} + 17 \cdot A_{i,j} + 58 \cdot A_{i,j+1} - 10 \cdot A_{i,j+2} + 4 \cdot A_{i,j+3} - A_{i,j+4}.
\end{aligned}$$

The actual implementation involves a right-shift by $(B - 8)$ bits after the above calculations, where $B \geq 8$ is the number of bits per image sample.

It should be obvious that the eight-tap hfilter is symmetric, so it works well for the half-pixel positions which are in the middle of pixels that are on the image grid. The seven-tap qfilter is asymmetric, well-suited for the quarter-pixel positions which are not in the middle. The subtly different treatment of a and c in Eqs. (12.9) and (12.11) reflects the nature of this asymmetric operation. Basically, qfilter$[1 - t]$ is a flipped version of qfilter$[t]$. For example, $a_{i,j}$ is closest to $A_{i.j}$, and it will draw the most from $A_{i,j}$ with the weight 58; whereas $a_{i,j}$ will draw the most from $A_{i,j+1}$ with the weight 58.

Similarly, the values at positions d, e, and f can be derived using the following:

$$d_{i,j} = \sum_{t=-3}^{3} A_{i+t,j} \cdot \text{qfilter}[t], \tag{12.12}$$

$$e_{i,j} = \sum_{t=-3}^{4} A_{i+t,j} \cdot \text{hfilter}[t], \tag{12.13}$$

$$f_{i,j} = \sum_{t=-2}^{4} A_{i+t,j} \cdot \text{qfilter}[1 - t]. \tag{12.14}$$

The other subpixel samples can be obtained from the vertically nearby a, b, or c pixels as below. To enable 16-bit operations, a right shift of six bits is introduced.

$$g_{i,j} = \left(\sum_{t=-3}^{3} a_{i+t,j} \cdot \text{qfilter}[t] \right) \gg 6,$$

$$j_{i,j} = \left(\sum_{t=-3}^{4} a_{i+t,j} \cdot \text{hfilter}[t] \right) \gg 6,$$

$$n_{i,j} = \left(\sum_{t=-2}^{4} a_{i+t,j} \cdot \text{qfilter}[1 - t] \right) \gg 6,$$

$$h_{i,j} = \left(\sum_{t=-3}^{3} b_{i+t,j} \cdot \text{qfilter}[t] \right) \gg 6,$$

$$k_{i,j} = \left(\sum_{t=-3}^{4} b_{i+t,j} \cdot \text{hfilter}[t] \right) \gg 6,$$

$$p_{i,j} = \left(\sum_{t=-2}^{4} b_{i+t,j} \cdot \text{qfilter}[1 - t] \right) \gg 6,$$

$$i_{i,j} = \left(\sum_{t=-3}^{3} c_{i+t,j} \cdot \text{qfilter}[t] \right) \gg 6,$$

$$m_{i,j} = \left(\sum_{t=-3}^{4} c_{i+t,j} \cdot \text{hfilter}[t] \right) \gg 6,$$

$$q_{i,j} = \left(\sum_{t=-2}^{4} c_{i+t,j} \cdot \text{qfilter}[1-t] \right) \gg 6.$$

12.2.2 Integer Transform

As in H.264, transform coding is applied to the prediction error residuals. The 2-D transform is accomplished by applying a 1-D transform in the vertical and then horizontal direction. This is implemented by two matrix multiplications: $\mathbf{F} = \mathbf{H} \times \mathbf{f} \times \mathbf{H}^T$, where $\mathbf{f}$ is the input residual data and $\mathbf{F}$ is the transformed data. $\mathbf{H}$ is the Integer Transform matrix that approximates the DCT-matrix.

Transform block sizes of 4×4, 8×8, 16×16, and 32×32 are supported. Only one Integer Transform Matrix, i.e., $\mathbf{H_{32\times 32}}$ is specified in H.265. The other matrices for smaller TBs are subsampled versions of $\mathbf{H_{32\times 32}}$. For example, $\mathbf{H_{16\times 16}}$ shown below is for the 16×16 TBs.

$$\mathbf{H_{16\times 16}} = \begin{bmatrix}
64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 \\
90 & 87 & 80 & 70 & 57 & 43 & 25 & 9 & -9 & -25 & -43 & -57 & -70 & -80 & -87 & -90 \\
89 & 75 & 50 & 18 & -18 & -50 & -75 & -89 & -89 & -75 & -50 & -18 & 18 & 50 & 75 & 89 \\
87 & 57 & 9 & -43 & -80 & -90 & -70 & -25 & 25 & 70 & 90 & 80 & 43 & -9 & -57 & -87 \\
83 & 36 & -36 & -83 & -83 & -36 & 36 & 83 & 83 & 36 & -36 & -83 & -83 & -36 & 36 & 83 \\
80 & 9 & -70 & -87 & -25 & 57 & 90 & 43 & -43 & -90 & -57 & 25 & 87 & 70 & -9 & -80 \\
75 & -18 & -89 & -50 & 50 & 89 & 18 & -75 & -75 & 18 & 89 & 50 & -50 & -89 & -18 & 75 \\
70 & -43 & -87 & 9 & 90 & 25 & -80 & -57 & 57 & 80 & -25 & -90 & -9 & 87 & 43 & -70 \\
64 & -64 & -64 & 64 & 64 & -64 & -64 & 64 & 64 & -64 & -64 & 64 & 64 & -64 & -64 & 64 \\
57 & -80 & -25 & 90 & -9 & -87 & 43 & 70 & -70 & -43 & 87 & 9 & -90 & 25 & 80 & -57 \\
50 & -89 & 18 & 75 & -75 & -18 & 89 & -50 & -50 & 89 & -18 & -75 & 75 & 18 & -89 & 50 \\
43 & -90 & 57 & 25 & -87 & 70 & 9 & -80 & 80 & -9 & -70 & 87 & -25 & -57 & 90 & -43 \\
36 & -83 & 83 & -36 & -36 & 83 & -83 & 36 & 36 & -83 & 83 & -36 & -36 & 83 & -83 & 36 \\
25 & -70 & 90 & -80 & 43 & 9 & -57 & 87 & -87 & 57 & -9 & -43 & 80 & -90 & 70 & -25 \\
18 & -50 & 75 & -89 & 89 & -75 & 50 & -18 & -18 & 50 & -75 & 89 & -89 & 75 & -50 & 18 \\
9 & -25 & 43 & -57 & 70 & -80 & 87 & -90 & 90 & -87 & 80 & -70 & 57 & -43 & 25 & -9
\end{bmatrix} \tag{12.15}$$

$\mathbf{H_{8\times 8}}$ can be obtained by using the first 8 entries of Rows 0, 2, 4, 6, ... of $\mathbf{H_{16\times 16}}$. For $\mathbf{H_{4\times 4}}$, use the first 4 entries of Rows 0, 4, 8, and 12.

$$\mathbf{H_{4\times 4}} = \begin{bmatrix} 64 & 64 & 64 & 64 \\ 83 & 36 & -36 & -83 \\ 64 & -64 & -64 & 64 \\ 36 & -83 & 83 & -36 \end{bmatrix} \tag{12.16}$$

Compared to Eq. (12.2), the entries in $\mathbf{H_{4\times 4}}$ clearly have much larger magnitudes. In order to use 16-bit arithmetic and 16-bit memory, the dynamic range of the intermediate results from the first matrix multiplication must be reduced by introducing a 7-bit right shift and 16-bit clipping operation.

12.2.3 Quantization and Scaling

Unlike the **H** matrix in H.264 (Eq. 12.2), the numbers in the H.265 integer transform matrices, e.g., Eq. (12.15), are proportionally very close to the actual values of the DCT basis functions. Hence, the ad hoc scaling factors as built in Tables 12.1 and 12.2 are no longer needed.

For quantization, the quantization matrix and the same parameter QP as in H.264 are employed. The range of QP is [0, 51]. Similarly, the quantization step size doubles when the QP value is increased by 6.

12.2.4 Intra Coding

As in H.264, spatial predictions are used in intra coding in H.265. The neighboring boundary samples from the blocks at the top and/or left of the current block are used for the predictions. The prediction errors are then sent for transform coding. The transform block (TB) size ranges from 4×4 to 32×32 in Intra coding in H.265. Because of (a) the potentially much larger TB size, and (b) the effort to reduce prediction errors, the possible number of prediction modes is increased from 9 in H.264 to 35 in H.265. As shown in Fig. 12.16a, Mode 2 to Mode 34 are Intra_angular prediction modes. Note, the angle difference between modes is deliberately made uneven, e.g., to make it denser near horizontal or vertical directions. Most of the samples that are needed for angular predictions will be at subpixel positions. Bilinear interpolation of the two nearest pixels at integer positions is employed, and the precision is up to 1/32 pixel.

The two special prediction modes are Mode 0: Intra_Planar and Mode 1: Intra_DC. They are similar as in H.264. In Intra_DC, the average of the reference samples is used as the prediction. In Intra_planar, different from H.264, all four corners are used for the planar prediction, i.e., two plane predictions will be made and the average of their values will be adopted.

12.2.5 Discrete Sine Transform

In Intra_4×4, for luma residual blocks, HEVC introduced an alternative transform based on one of the variants of the *Discrete Sine Transform (DST)* (the so-called DST-VII) [15]. It is because the intra predictions are based on the neighboring boundary samples on the top or at the left of the block. The prediction error tends to increase for the nodes in the block that are farther away from the top or left neighboring samples. In general, DST is found to cope with this situation better than DCT at the transform coding step.

The integer matrix for DST can be described by:

$$\mathbf{H_{DST}}[i, j] = \text{round}\left(128 \times \frac{2}{\sqrt{2N+1}} \sin\frac{(2i-1)j\pi}{2N+1}\right), \tag{12.17}$$

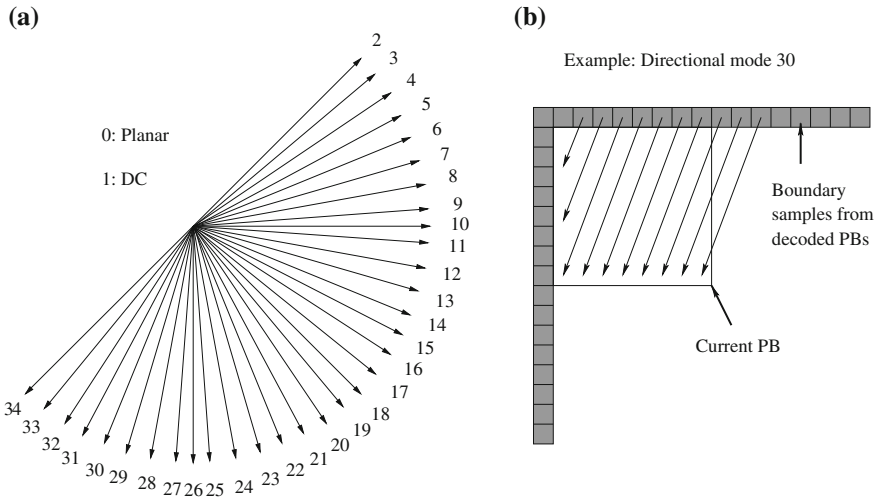

Fig. 12.16 H.265 intra prediction. **a** Modes and intra prediction directions, **b** Intra prediction for an 8×8 block

where $i = 1, .., N$ and $j = 1, .., N$ are the row and column indices, and the block size is $N \times N$.

When $N = 4$, the following $\mathbf{H_{DST}}$ is obtained:

$$\mathbf{H_{DST}} = \begin{bmatrix} 29 & 55 & 74 & 84 \\ 74 & 74 & 0 & -74 \\ 84 & -29 & -74 & 55 \\ 55 & -84 & 74 & -29 \end{bmatrix} \tag{12.18}$$

Saxena and Fernandes [16,17] further studied the benefit of combining DCT and DST, i.e., allowing either DCT or DST in one of the two 1-D transforms, because DST and DCT are shown to win in either the vertical and/or horizontal direction(s) for certain prediction modes. Although there are over 30 different intra prediction directions in H.265, they classify the prediction modes into:

- Category 1—the samples for prediction are either all from the left neighbors of the current block (Fig. 12.17a), or all from the top neighbors of the current block (Fig. 12.17b).
- Category 2—the samples for prediction are from both the top and left neighbors of the current block (Fig. 12.17c, d).
- DC—a special prediction mode in which the average of a fixed set of neighboring samples is used.

Table 12.8 shows some of their recommendations.

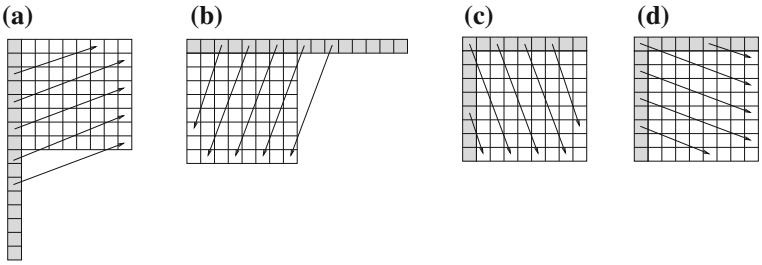

Fig. 12.17 Intra prediction directions in H.265. **a** Category 1, predictions from *left* neighbors only, **b** Category 1, predictions from *top* neighbors only, **c** and **d** Category 2, predictions from both *top* and *left* neighbors

Table 12.8 Combining DCT and DST for intra coding

Intra prediction Category	Neighboring samples used	Vertical (column) transform	Horizontal (row) transform
Category 1	from left only	DCT	DST
Category 1	from top only	DST	DCT
Category 2	from both top and left	DST	DST
DC	special (from a fixed set)	DCT	DCT

12.2.6 In-Loop Filtering

Similar to H.264, in-loop filtering processes are applied in order to remove the blocky artifacts. In addition to Deblocking Filtering, H.265 also introduces a *Sample Adaptive Offset (SAO)* process.

Deblocking Filtering

Instead of applying deblocking filtering to 4×4 blocks as in H.264, it is applied only to edges that are on the 8×8 image grid. This reduces the computation complexity, and it is especially good for parallel processing since the chance of cascading changes at nearby samples is greatly reduced. The visual quality is still good, partly due to the SAO process described below.

The deblocking filtering is applied first to the vertical edges, then to the horizontal edges in the picture, thus enabling parallel processing. Alternatively, it can be applied CTB by CTB.

Sample Adaptive Offset (SAO)

The SAO process can be invoked optionally after the deblocking filtering. Basically, an offset value is added to each sample based on certain conditions described below.

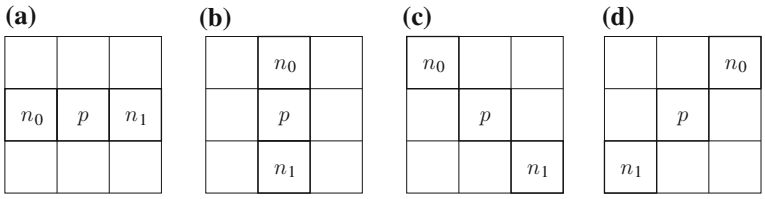

Fig. 12.18 Neighboring samples considered in SAO edge offset mode

Two modes are defined for applying the SAO: *Band offset mode* and *Edge offset mode*.

In the Band offset mode, the range of the sample amplitudes is split into 32 bands. A band offset can be added to the sample values in four of the consecutive bands simultaneously. This helps to reduce the "banding artifacts" in smooth areas.

In the Edge offset mode, the gradient (edge) information is analyzed first. Figure 12.18 depicts the four possible gradient (edge) directions: (a) horizontal, (b) vertical, and (c, d) diagonals. A positive or negative offset, or zero offset can be added to the sample p based on the following:

- Positive: p is a local minimum ($p < n_0$ & $p < n_1$), or p is an edge pixel ($p < n_0$ & $p = n_1$ or $p = n_0$ & $p < n_1$).
- Negative: p is a local maximum ($p > n_0$ & $p > n_1$), or p is an edge pixel ($p > n_0$ & $p = n_1$ or $p = n_0$ & $p > n_1$).
- Zero: None of the above.

12.2.7 Entropy Coding

H.265 only uses CABAC in entropy coding, i.e., CAVLC is no longer used. Because of the newly introduced coding tree and transform tree structure, the tree depth now becomes an important part of the *context modeling* in addition to the spatially neighboring context in H.264/AVC. As a result, the number of contexts is reduced, and the entropy coding efficiency is further improved.

Unlike previous video standards, three simple scanning methods are defined to read in the transform coefficients, i.e., *Diagonal up-right, Horizontal*, and *Vertical*. The goal is still to maximize the length of zero-runs. The scanning always takes place in 4×4 subblocks regardless of the TB size. The diagonal up-right scan is used for all inter-predicted blocks and for intra-predicted blocks that are 16×16 or 32×32. For intra-predicted blocks that are 4×4 or 8×8, the following are used: Horizontal—for prediction directions close to vertical, Vertical—for prediction directions close to horizontal, Diagonal up-right—for the other prediction directions.

There are many improvements as how to code the nonzero transform coefficients efficiently [12,14]. Also, one of the goals of the new implementation of CABAC in H.265 is to simplify its context representations so its throughput can be increased. For details readers are referred to [18] which provides an excellent reference.

Table 12.9 Sample video formats supported in the H.265 main profile

Level(s)	Max Luma picture width × height	Max Luma picture size (samples)	Frame rate (fps)	Main tier max bitrate (Mb/s)
1	176 × 144	36864	15	0.128
2	352 × 288	122880	30	1.5
2.1	640 × 360	245760	30	3.0
3	960 × 540	552960	30	6.0
3.1	1280 × 720	983040	30	10
4 / 4.1	2048 × 1080	2228224	30 / 60	12 / 20
5 / 5.1 / 5.2	4096 × 2160	8912896	30 / 60 / 120	25 / 40 / 60
6 / 6.1 / 6.2	8192 × 4320	35651584	30 / 60 / 120	60 / 120 / 240

12.2.8 Special Coding Modes

Three special coding modes are defined in H.265. They can be applied at the CU or TU level.

- **I_PCM** As in H.264, the prediction, transform coding, quantization, and entropy coding steps are bypassed. The PCM-coded (fixed-length) samples are sent directly. It is invoked when other prediction modes failed to produce any data reduction.
- **Lossless** The residual errors from inter- or intra-predictions are sent to entropy coding directly, thus to avoid any lossy steps, especially the quantization after transform coding.
- **Transform skipping** Only the transform step is bypassed. This works for certain data (e.g., computer-generated images or graphics). It can only be applied to 4×4 TBs.

12.2.9 H.265 Profiles

At this point, only three profiles are defined: *Main profile*, *Main 10 profile*, and *Main Still Picture profile*, although more and higher profiles are expected in the future.

The default format for color is YCbCr. In all main profiles, chroma subsampling is 4:2:0. Each sample has 8 bits, except in Main 10, which has 10 bits.

As an example, some of the video formats that are supported at various *levels* in the Main profile are listed in Table 12.9. As shown, the total number of levels proposed is 13. It covers very low resolution videos, e.g., QCIF (176×144) at Level 1, as well as very high resolution videos, e.g., UHDTV ($8,192 \times 4,320$) at Levels 6, 6.1, and 6.2.

In calculating the Max Luma Picture Size, the width and height are rounded up to the nearest multiples of 64 (as an implementation requirement). For example, 176×144 becomes $192 \times 192 = 36,864$.

Table 12.10 Average bitrate reductions under equal PSNR

Video Compression Method	H.264/MPEG-4 AVC HP (%)	MPEG-4 ASP (%)	MPEG-2/H.262 MP (%)
H.265 MP	35.4	63.7	70.8
H.264/MPEG-4 AVC HP	-	44.5	55.4
MPEG-4 ASP	-	-	19.7

As shown, the current HDTV is at Levels 4 and 4.1 for Frame Rates of 30 and 60 fps. The maximum bitrates of the compressed video at the so-called Main Tier are 12 Mbits/sec and 20 Mbits/sec, respectively. At the High Tier, they can be as much as 2.5 times higher. The UHDTV videos at Level 5 and above will demand much higher bitrates which remain as a challenge for all aspects of multimedia including storage, data transmission and display devices.

12.3 Comparisons of Video Coding Efficiency

When comparing the coding efficiency of different video compression methods, a common practice is to compare the bitrates of the coded video bitstreams at the same *quality*. The video quality assessment approaches can be *objective* or *subjective*: the former is done automatically by computers and the latter requires human judgment.

12.3.1 Objective Assessment

The most common criterion used for the objective assessment is peak signal-to-noise ratio (PSNR). As defined in Sect. 8.3. For images it is:

$$PSNR = 10 \log_{10} \frac{I_{max}^2}{MSE}, \tag{12.19}$$

where I_{max} is the maximum intensity value, e.g., 255 for 8-bit images, and MSE is the *Mean Squared Error* between the original image I and the compressed image $\tilde{I}$. For videos, the PSNR is usually the average of the PSNRs of the images in the video sequence.

Ohm et al. [19] reported many of their experimental results. As an example, Table 12.10 lists the average bitrate reductions when different video compression methods are compared at the same PSNR, in this case in the range 32–42 dB. The test data are entertainment videos which are generally of higher quality and have higher resolutions. The notations are: MP—Main Profile, HP—High Profile, ASP—Advanced Simple Profile. For example, when H.265 MP is compared with H264/MPEG-4 AVC HP, a saving of 35.4 % is realized.

12.3.2 Subjective Assessment

The main advantage of PSNR is that it is easy to calculate. However, it does not necessarily reflect the quality as perceived by humans, i.e., visual quality. An obvious example would be to add (or subtract) a small and fixed amount to the intensity values of all the pixels in the picture. Visually (subjectively), we may not notice any quality change. On the other hand, the PSNR will certainly be affected.

The methodology of subjective assessment of television pictures is specified in ITU-R Recommendation BT.500. Its latest version is BT.500-13 [20] revised in 2012.

In Ohm et al.'s experiment [19], the original and compressed video clips are shown in succession to the human subjects (the so-called *double stimulus* method). The subjects are asked to grade the videos by their quality, 0–lowest, 10–highest. The Mean Opinion Score (MOS), which is the arithmetic mean of their scores, is used as the measure for the subjective quality of different video compression methods.

It is reported that when compared with H.264/MPEG-4 AVC HP, at approximately the same subjective quality, for the nine test videos for entertainment applications, the average bitrate reductions by H.265 MP range from 29.8 to 66.6 %, with an average of 49.3 %. This is very close to the original goal of 50 % reduction.

Video Quality Assessment (VQA) is an active research area. The main efforts are to find better metrics than the simple measures such as PSNR, so the assessments can be conducted objectively (by computers) and their results will be comparable to those of human subjects. Wang et al. [21] presented the *Structural Similarity (SSIM)* index that captures some simple image structural information (e.g., luminance and contrast). It has become very popular in image and video quality assessments. Peng et al. [22] presented a brief survey of VQA and a good metric based on a novel spacetime texture representation.

12.4 Exercises

1. Integer Transforms are used in H.264 and H.265.

   (a) What is the relationship between the DCT and Integer Transform?

   (b) What are the main advantages of using Integer Transform instead of DCT?

2. H.264 and H.265 use quarter-pixel precision in motion compensation.

   (a) What is the main reason that subpixel (in this case quarter-pixel) precision is advocated?

   (b) How do H.264 and H.265 differ in obtaining at quarter-pixel positions?

3. From Eq. 12.15, derive $\mathbf{H_{8\times 8}}$ for the Integer Transform in H.265.

4. H.264 and H.265 support *in-loop deblocking filtering*.

   (a) Why is deblocking a good idea? What are its disadvantages?

(b) What are the main differences in its H.264 and H.265 implementations?

(c) Beside the deblocking filtering, what does H.265 do to improve the visual quality?

5. Name at least three features in H.265 that facilitate parallel processing.
6. Give at least three reasons to argue that PSNR is not necessarily a good metric for video quality assessment.
7. P-frame coding in H.264 uses *Integer Transform*. For this exercise, assume:

$$F(u, v) = H \cdot f(i, j) \cdot H^T, \text{ where } H = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix}.$$

(a) What are the two advantages of using Integer Transform?

(b) Assume the Target Frame below is a P-frame. For simplicity, assume the size of macroblock is 4×4. For the macroblock shown in the Target Frame:

(i) What should be the Motion Vector?

(ii) What are the values of $f(i, j)$ in this case?

(iii) Show all values of $F(u, v)$.

20	40	60	80	100	120	140	155
30	50	70	90	110	130	150	165
40	60	80	100	120	140	160	175
50	70	90	110	130	150	170	185
60	80	100	120	140	160	180	195
70	90	110	130	150	170	190	205
80	100	120	140	160	180	200	215
85	105	125	145	165	185	205	220

Reference Frame

110	132	154	176	–	–	–	–
120	142	164	186	–	–	–	–
130	152	174	196	–	–	–	–
140	162	184	206	–	–	–	–
–	–	–	–	–	–	–	–
–	–	–	–	–	–	–	–
–	–	–	–	–	–	–	–
–	–	–	–	–	–	–	–

Target Frame

8. Write a program for the kth Order Exp-Golomb encoder and decoder. (a) What is the EG_0 codeword for unsigned $N = 110$? (b) Given an EG_0 code 000000011010011, what is the decoded unsigned N? (c) What is the EG_3 codeword for unsigned $N = 110$?
9. Write a program to implement video compression with motion compensation, transform coding, and quantization for a simplified H.26* encoder and decoder.
   - Use 4:2:0 for chroma subsampling.
   - Choose a video frame sequence (I-, P-, B-frames) similar to MPEG-1, 2. No interlacing.
   - For I-frames, implement the H.264 Intra_4×4 predictive coding.
   - For P- and B-frames, use only 8×8 for motion estimation. Use logarithmic search for motion vectors. Afterwards, use the 4×4 Integer Transform as in H.264.

- Use the quantization and scaling matrices as specified in Eqs. 12.5 and 12.7. Control and show the effect of various levels of compression and quantization losses.
- Do not implement the entropy coding part. Optionally, you may include any publicly available code for this.

10. Write a program to verify the results in Table 12.8. For example, to show that DST will produce shorter code than DCT for Category 2 directional predictions.

References

1. T. Wiegand, G.J. Sullivan, G. Bjøntegaard, A. Luthra, Overview of the H.264/AVC video coding standard. IEEE Trans. Circ. Syst. Video Technol. **13**(7), 560–576 (2003)
2. ITU-T H.264—ISO/IEC 14496–10. Advanced video coding for generic audio-visual services. ITU-T and ISO/IEC (2009)
3. ISO/IEC 14496, Part 10. *Information Technology: Coding of Audio-Visual Objects* (Part 10: Advanced Video Coding). ISO/IEC (2012)
4. I.E. Richardson, *The H.264 Advanced Video Compression Standard*, 2nd edn. (Wiley, 2010)
5. H.S. Malvar et al., Low-complexity transform and quantization in H.264/AVC. IEEE Trans. Circ. Syst. Video Technol. **13**(7), 598–603 (2003)
6. G. Bjontegaard, K. Lillevold, *Context-Adaptive VLC Coding of Coefficients*. JVT document JVT-C028 (2002)
7. I.E. Richardson, *H.264 and MPEG-4 Video Compression*. (Wiley, 2003)
8. D. Marpe, H. Schwarz, T. Wiegand, Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard. IEEE Trans. Circ. Syst. Video Technol. **13**(7), 620–636 (2003)
9. D. Marpe, T. Wiegand, The H.264/MPEG4 advanced video coding standard and its applications. IEEE Commun. Mag. **44**(8), 134–143 (2006)
10. H. Schwarz et al., Overview of scalable video coding extension of the H.264/AVC standard. IEEE Trans. Circ. Syst. Video Technol. **17**(9), 1103–1120 (2007)
11. P. Merkle et al., Efficient prediction structures for multiview video coding. IEEE Trans. Circ. Syst. Video Technol. **17**(11), 1461–1473 (2007)
12. G.J. Sullivan et al., Overview of the high efficiency video coding (HEVC) standard. IEEE Trans. Circ. Syst. Video Technol. **22**(12), 1649–1668 (2012)
13. J.R. Ohm, G.J. Sullivan, High efficiency video coding: the next frontier in video compression. IEEE Signal Process. Mag. **30**(1), 152–158 (2013)
14. ITU-T H.265—ISO/IEC 23008–2. H.265: high efficiency video coding. ITU-T and ISO/IEC (2013)
15. R.K. Chivukula, Y.A. Reznik, Fast computing of discrete cosine and sine transforms of types VI and VII. Proc. SPIE (Applications of digital image processing XXXIV) **8135**, 813505 (2011)
16. A. Saxena, F.C. Fernandes, Mode dependent DCT/DST for intra prediction in block-based image/video coding. IEEE Int. Conf. Image Process. pp. 1685–1688 (2011)
17. A. Saxena, F.C. Fernandes, DCT/DST based transform coding for intra prediction in image/video coding. IEEE Trans. Image Process. **22**(10), 3974–3981 (2013)
18. V. Sze, M. Budagavi, High throughput CABAC entropy coding in HEVC. IEEE Trans. Circ. Syst. Video Technol. **22**(12), 1778–1791 (2012)

19. J.R. Ohm et al., Comparison of the coding efficiency of video coding standards: including high efficiency video coding (HEVC). IEEE Trans. Circ. Syst. Video Technol. **22**(12), 1669–1684 (2012)
20. ITU-R Rec. BT.500-13. Methodology for the subjective assessment of the quality of television pictures. ITU-R (2012)
21. Z. Wang et al., Image quality assessment: from error visibility to structural similarity. IEEE Trans. Image Process. **13**(4), 600–612 (2004)
22. P. Peng, K. Cannons, Z.N. Li, in *ACM MM'13: Proceedings of the ACM International Conference on Multimedia*. Efficient video quality assessment based on spacetime texture representation (2013)