CHAPTER

9 The Block-Based Hybrid Video Codec

CHAPTER OUTLINE

In Chapter 8 we saw how motion estimation formed the basis of an efficient video coding system. This chapter describes how it can be integrated into a practical video compression framework based on a hybrid structure, combining block-based

For colour and a higher quality version of Figure 9.17 please refer to the electronic version or the website.

Communicating Pictures. http://dx.doi.org/10.1016/B978-0-12-405906-1.00009-X

motion prediction with block-based transform coding. The architecture of a generic video encoder is presented and the operation of the encoding and decoding stages is described in detail.

The limitations of the basic hybrid architecture are discussed and a collection of practical codec enhancements that have been introduced into recent standards are described. These techniques serve to significantly enhance the performance of the basic video coding structure and have enabled the basic hybrid architecture to provide a doubling of coding gain every 8–10 years over the past 40 years or so. The techniques described are not an exhaustive list of all the features found in modern codecs, but are some of the most significant ones.

We first of all focus on intra-prediction—a means of enhancing performance through spatial prediction of blocks in each frame. We then extend the motion prediction methods introduced in Chapter 8 through sub-pixel estimation and the use of multiple reference frames. We also examine the influence of variable block sizes and their exploitation in the coding process, both in terms of transforms and motion estimation. Finally we study the important area of in-loop deblocking as an approach to reducing edge artifacts.

9.1 The block-based hybrid model for video compression

9.1.1 Picture types and prediction modes

Prediction modes

As we saw in Chapter 8, three main classes of prediction are used in video compression:

- **Forward prediction:** The reference picture occurs temporally before the current picture.
- **Backward prediction:** The reference picture occurs temporally after the current picture.
- **Bidirectional prediction:** Two (or more) reference pictures (forward and backward) are employed and the candidate predictions are combined in some way to form the final prediction.

Picture types

Three major types of picture (or frame) are employed in most video codecs:

- **I-pictures:** These are intra-coded (coded without reference to any other pictures).
- **P-pictures:** These are inter-coded with forward (or backward) prediction from another I- or P-picture.
- **B-pictures:** These are inter-coded with bidirectional prediction from more than one I- and/or P-picture.

Coded pictures are arranged in a sequence known as a *Group of Pictures* (GOP). A typical GOP structure comprising 12 frames is shown in Figure 9.1. A GOP will

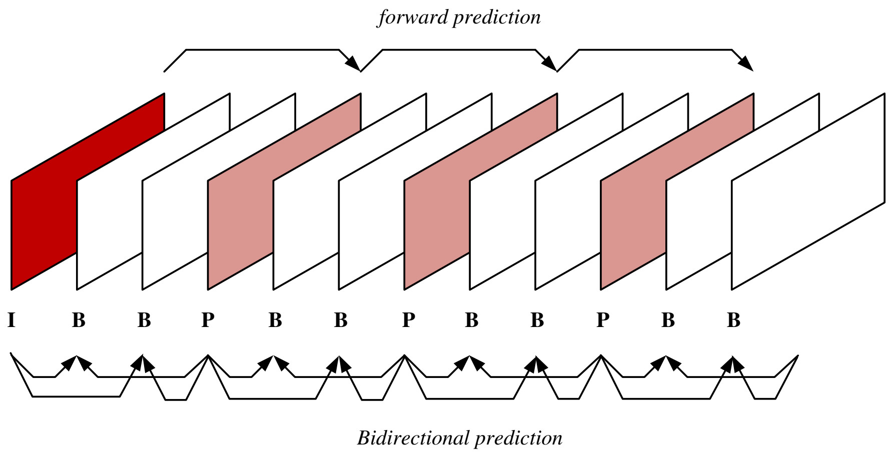


FIGURE 9.1

Typical group of pictures structure.

contain one I-picture and zero or more P- and B-pictures. The 12-frame GOP in Figure 9.1 is sometimes referred to as an IBBPBBPBBPBB structure and it is clear, for reasons of causality, that the encoding order is different to that shown in the figure since the P-pictures must be encoded prior to the preceding B-pictures.

9.1.2 Properties of the DFD signal

The DCT is a special case of the Karhunen–Loeve expansion for stationary and first order Markov processes whose autocorrelation coefficient approaches unity. Most still images possess good correlation properties, typically with $\rho \geq 0.9$; this justifies the use of the DCT for intra-frame coding as it will provide close to optimum decorrelation. However, in the case of inter-frame coding, where motion is predicted through motion estimation in a DPCM loop, the autocorrelation coefficient for the DFD signal typically falls to 0.3–0.5. Strobach [1] showed that, in many practical cases, the DCT coding gain for a DFD signal is small compared to the intra-frame case and that most of the gain is associated with temporal decorrelation. The edges introduced in the DFD signal due to motion failure can also cause ringing artifacts after quantization.

This discussion implies that the hybrid codec, which uses a decorrelating transform such as the DCT to code the DFD residual, is not always optimum. In practice, however, the DFD can be made smoother (i.e. exhibit higher autocorrelation values) if the motion model is more accurate and if any artificial edges are filtered out. These characteristics are promoted by codec enhancements such as sub-pixel motion estimation, variable block sizes, multiple reference frames, and the use of a deblocking filter in the motion prediction loop. Such enhancements, alongside efficient rate–distortion optimization methods, are largely responsible for the performance, and universal acceptance, of the block-based hybrid codec. They are all discussed in more detail later in this chapter.

9.1.3 Operation of the video encoding loop

Following the brief overview of the video encoding architecture given in Chapter 1, we describe here the operation of the encoding loop more formally and in more detail. A generic structure is given in Figure 9.2 for the case of intra-frame coding, and in Figure 9.3 for the inter-frame mode. The operation of the encoder in intra-mode is

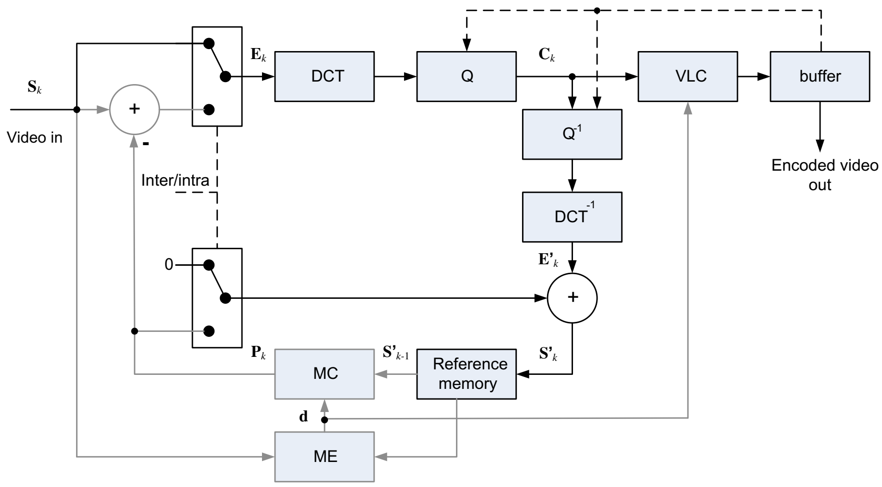


FIGURE 9.2
Video encoder structure (intra-frame mode). Dotted lines depict control signals relating to quantizer step size, etc.

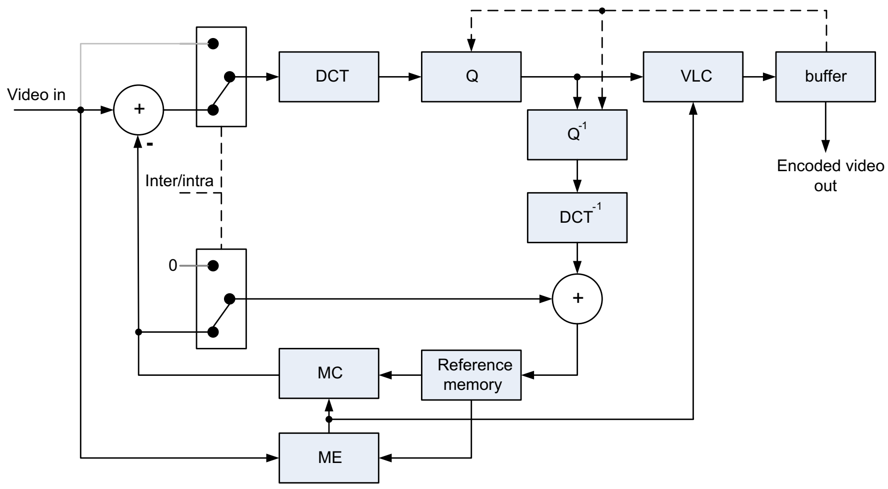


FIGURE 9.3
Video encoder structure (inter-frame mode).

Algorithm 9.1 Intra-mode encoder operation.

```
1. Inter/intra switch is in the intra position;
2. Compute displaced frame difference (DFD) signal (equal to the video input frame
   in this case): $\mathbf{E}_k = \mathbf{S}_k$;
3. Perform a forward decorrelating transform (typically a DCT or variant) on the
   input frame and quantize according to the prevailing rate–distortion criteria: $\mathbf{C}_k = \mathrm{Q}\left(\mathrm{DCT}\left(\mathbf{E}_k\right)\right)$;
4. Entropy code the transformed frame and transmit to the channel;
5. Inverse quantize $\mathbf{C}_k$ and perform an inverse DCT to produce the same decoded
   frame pixel values as at the decoder: $\mathbf{E}_k' = \mathrm{DCT}^{-1}\left(\mathrm{Q}^{-1}\left(\mathbf{C}_k\right)\right)$;
6. Update the reference memory with the reconstructed frame: $\mathbf{S}_k' = \mathbf{E}_k' + \mathbf{0}$.
```

Algorithm 9.2 Inter-mode encoder operation.

```
1. Inter/intra switch is in the inter position;
2. Estimate motion vector for current frame: $\mathbf{d} = \mathrm{ME}\left(\mathbf{S}_k, \mathbf{S}_{k-1}\right)$;
3. Form the motion compensated prediction frame, $\mathbf{P}_k$: $\mathbf{P}_k = \mathbf{S}_{k-1}'\left[\mathbf{p} + \mathbf{d}\right]$;
4. Compute the DFD signal: $\mathbf{E}_k = \mathbf{S}_k - \mathbf{P}_k$;
5. Perform a forward decorrelating transform on the DFD and quantize according to
   the prevailing rate–distortion criteria: $\mathbf{C}_k = \mathrm{Q}\left(\mathrm{DCT}\left(\mathbf{E}_k\right)\right)$;
6. Entropy code the transformed DFD, motion vectors and control parameters, and
   transmit to the channel;
7. Inverse quantize $\mathbf{C}_k$ and perform an inverse DCT to produce the same decoded
   frame pixel values as at the decoder: $\mathbf{E}_k' = \mathrm{DCT}^{-1}\left(\mathrm{Q}^{-1}\left(\mathbf{C}_k\right)\right)$;
8. Update the reference memory with the reconstructed frame: $\mathbf{S}_k' = \mathbf{E}_k' + \mathbf{P}_k$.
```

described in Algorithm 9.1. Similarly the operation in inter-mode is described in Algorithm 9.2.

9.1.4 Operation of the video decoder

The operation of the video decoder is illustrated in Figure 9.4 and described formally in Algorithms 9.3 and 9.4. We can see, by comparing the encoder and decoder architectures, that the encoder contains a complete replica of the decoder in its prediction feedback loop. This ensures that (in the absence of channel errors) there is no drift between the encoder and decoder operations.

Example 9.1 (Operation of a basic video encoder)
Based on the encoder diagrams in Figures 9.1 and 9.3 together with the following data at time k, compute:

1. The current integer-pixel motion vector, $\mathbf{d}$, at time k. Assume the search window is the whole reference frame and that vectors are restricted to point within the frame.

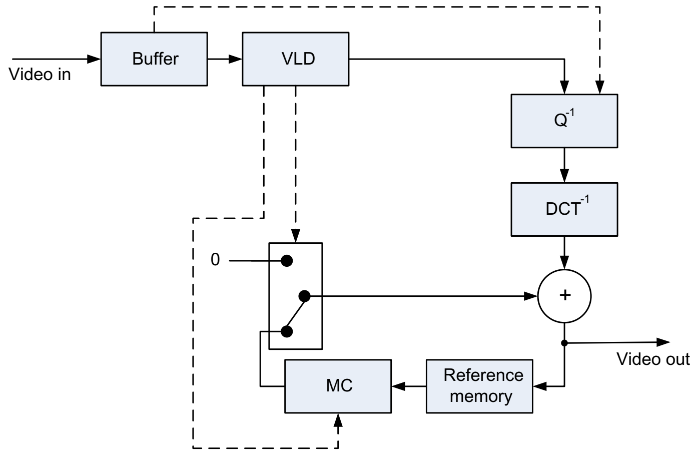


FIGURE 9.4

Video decoder structure.

Algorithm 9.3 Intra-mode decoder operation.

1. Inter/intra switch is in the intra position;
2. Perform entropy decoding of control parameters and quantized DFD coefficients;
3. Inverse quantize $\mathbf{C}_k$ and perform an inverse DCT to produce the decoded frame pixel values: $\mathbf{E}_k^{'} = \mathrm{DCT}^{-1}\left(\mathrm{Q}^{-1}\left(\mathbf{C}_k\right)\right)$;
4. Update the reference memory with the reconstructed frame and output to file or display: $\mathbf{S}_k^{'} = \mathbf{E}_k^{'} + \mathbf{0}$.

Algorithm 9.4 Inter-mode decoder operation.

1. Inter/intra switch is in the inter position;
2. Perform entropy decoding of control parameters, quantized DFD coefficients and motion vector;
3. Inverse quantize $\mathbf{C}_k$ and perform an inverse DCT to produce the decoded DFD pixel values: $\mathbf{E}_k^{'} = \mathrm{DCT}^{-1}\left(\mathrm{Q}^{-1}\left(\mathbf{C}_k\right)\right)$;
4. Form the motion compensated prediction frame, $\mathbf{P}_k$: $\mathbf{P}_k = \mathbf{S}_{k-1}^{'}\left[\mathbf{p}+\mathbf{d}\right]$;
5. Update the reference memory with the reconstructed frame and output to file or display: $\mathbf{S}_k^{'} = \mathbf{E}_k^{'} + \mathbf{P}_k$.

2. The motion compensated output, $\mathbf{P}_k$, at time k.
3. The DFD for the current input frame, $\mathbf{E}_k$.
4. The transformed and quantized DFD output, $\mathbf{C}_k$.

Assume an image frame size of 12×12 pixels and a macroblock size of 4×4 pixels. The codec is operating in its inter-frame mode, using block-based translational motion estimation with a block size of 4×4 pixels:

Transform matrix: $\mathbf{A} = \frac{1}{2}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix}$

Quantization matrix: $\mathbf{Q} = \begin{bmatrix} 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 \\ 2 & 2 & 4 & 4 \\ 2 & 2 & 2 & 4 \end{bmatrix}$

Current input block: $\mathbf{S}_k = \begin{bmatrix} 6 & 6 & 6 & 6 \\ 8 & 8 & 8 & 8 \\ 6 & 6 & 6 & 6 \\ 8 & 8 & 8 & 8 \end{bmatrix}$; assume top left of frame

Reference memory: $\mathbf{S}'_{k-1} = \begin{bmatrix} 0 & 0 & 5 & 6 & 2 & 6 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 8 & 8 & 8 & 8 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 6 & 6 & 6 & 6 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 8 & 8 & 12 & 8 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 7 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 4 & 2 & 3 & 4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 4 & 5 & 8 & 0 \\ 7 & 7 & 8 & 9 & 0 & 0 & 0 & 0 & 6 & 7 & 0 & 0 \\ 0 & 0 & 0 & 10 & 3 & 0 & 0 & 0 & 0 & 7 & 0 & 0 \\ 0 & 0 & 2 & 2 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$

Solution.

1. Current motion vector:
   By inspection:

$$\mathbf{d} = [2, 0]$$

2. Motion compensated output:

$$\mathbf{P}_k = \begin{bmatrix} 5 & 6 & 2 & 6 \\ 8 & 8 & 8 & 8 \\ 6 & 6 & 6 & 6 \\ 8 & 8 & 12 & 8 \end{bmatrix}$$

3. DFD for current input frame:

$$\mathbf{E}_k = \mathbf{S}_k - \mathbf{P}_k = \begin{bmatrix} 1 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -4 & 0 \end{bmatrix}$$

4. Transformed and quantized output:
The transformed output is:

$$\mathbf{C}_k = \mathbf{A}\mathbf{E}_k\mathbf{A}^{\mathrm{T}} = \frac{1}{4}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 9 & -7 & -7 & 9 \\ 1 & 1 & 1 & 1 \\ 9 & -7 & -7 & 9 \end{bmatrix}$$

and the quantized output is:

$$\mathbf{C}_{k(\mathrm{Q})} = \frac{1}{4}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 9 & -7 & -7 & 9 \\ 1 & 1 & 1 & 1 \\ 9 & -7 & -7 & 9 \end{bmatrix} / \mathbf{Q} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & -1 & -1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

9.2 Intra-frame prediction

In Chapter 8 we saw the benefits of temporal prediction as a means of decorrelating between frames prior to transformation. H.264/AVC and HEVC also support intra-prediction in the spatial domain as well as an I-PCM mode that enables direct encoding of pixel values. Intra-coding has been found to be especially beneficial in regions which have certain oriented textures or directional edges. There are two main prediction modes supported in H.264/AVC—intra_4 × 4 and intra_16 × 16. These are described below.

9.2.1 Intra-prediction for small luminance blocks

This mode performs well when there is a high level of local detail, particularly with oriented features such as edges. Coding of a given block (4 × 4 in H.264/AVC) is performed with reference to a subset of samples from previously coded blocks that lie to the left and above the current block. As we will see in Chapter 11, intra-coding can lead to the propagation of errors between frames when it is based on data predicted from previously inter-coded macroblocks. To avoid this a *constrained intra-coding mode* can be used that restricts intra-prediction only to be based on intra-coded neighbors.

Figure 9.5 shows, for example, the intra-coding relationships and orientations used in H.264/AVC. The prediction for each 4 × 4 block (pixels *a*–*p* here) is based on its neighboring pixels (*A*–*M*). In H.264/AVC, the predictor can select from nine

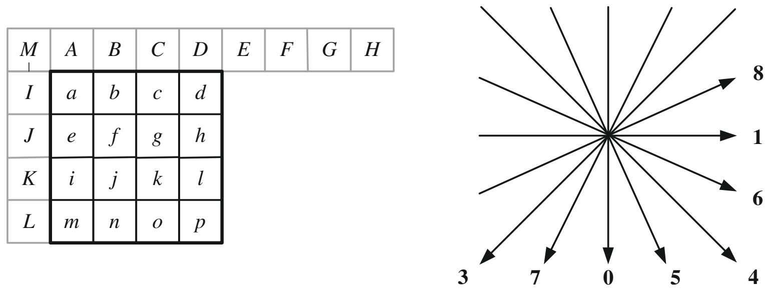


FIGURE 9.5

Intra-prediction modes in H.264/AVC.

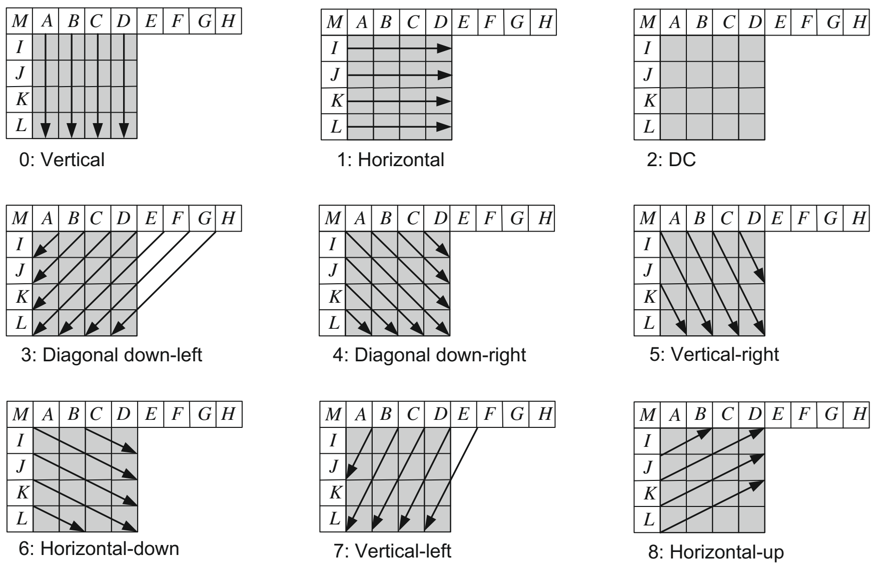


FIGURE 9.6

4 × 4 intra-prediction modes.

prediction orientations plus a DC mode where the average of the neighboring pixels is used as the prediction. These nine modes are shown in Figure 9.6. Some examples of how the predictions are computed are given below:

Mode 0 (Vertical)

$$\{a, e, i, m\} = A \tag{9.1}$$

Mode 1 (Horizontal)

$$\{a, b, c, d\} = I \tag{9.2}$$

Mode 2 (DC)

$$\{a, b, \cdots, p\} = \left\lfloor \left(\frac{A + B + C + D + I + J + K + L}{8} \right) + 0.5 \right\rfloor \tag{9.3}$$

Mode 4 (Down-right)

$$\{a, f, k, p\} = \left\lfloor \left(\frac{A + 2M + I}{4} \right) + 0.5 \right\rfloor \tag{9.4}$$

Other modes are formed by a similar extrapolation approach and a full description can be found in Ref. [2].

9.2.2 Intra-prediction for larger blocks

For larger smoother image regions it can be beneficial to perform intra-prediction on larger blocks. In the H.264 intra_16 $\times$ 16 mode, the entire 16 $\times$ 16 luma block is predicted simultaneously. The mode supports only four prediction types: mode 0 (vertical), mode 1 (horizontal), mode 2 (DC), and mode 3 (plane). The first three modes are similar to the 4 $\times$ 4 modes described above except they reference 16 pixels above and to the left of the current block, rather than just four. Details of the plane mode can be found in Ref. [2].

Chroma samples can be predicted in a similar manner to that used for luma blocks. Since chroma signals normally exhibit smoother characteristics than luma signals, only larger block sizes tend to be used.

Example 9.2 (Intra-prediction)
Compute the SAD values using the intra_4 $\times$ 4 vertical, horizontal, and DC modes, for the 4 $\times$ 4 block of pixels shown below:

1	2	3	4	5	6	7	8	9
1	2	3	4	5				
1	2	2	4	6				
2	2	3	4	3				
2	2	3	4	5				

Solution. The prediction blocks based on vertical, horizontal, and DC modes are given in the figure below:

2	3	4	5
2	3	4	5
2	3	4	5
2	3	4	5

vertical

1	1	1	1
1	1	1	1
2	2	2	2
2	2	2	2

horizontal

3	3	3	3
3	3	3	3
3	3	3	3
3	3	3	3

DC

and the corresponding SAD values are: SAD(vertical) = 4; SAD(horizontal) = 30; SAD(DC) = 16. Hence, based on this limited subset, the best match in terms of prediction residual energy is mode 0 (vertical). In this case the residual prior to coding would be as shown below:

0	0	0	0
0	–1	0	1
0	0	0	–2
0	0	0	0

9.3 Sub-pixel motion estimation

9.3.1 Sub-pixel matching

In Chapter 8 we introduced the concepts and realization of motion estimation as the basis for efficient video encoding. We limited the resolution of our search and matching to that of the sampling grid used to acquire the images. In reality, the motion in a scene is not related to the sampling grid and the most accurate match might occur with a sub-pixel displacement. Ericsson [3] introduced the concept of sub-pixel estimation in 1985 demonstrating, for the sequences used, that up to 2 dB coding gain could be achieved when changing from full-pixel to 1/8 pixel accuracy. This was formalized by Girod [4] who showed that limited gains were achieved by decreasing the estimation accuracy beyond 1/8 pixel resolution.

It is clear that the benefits of sub-pixel estimation will depend on a number of factors, including video format (spatial and temporal resolution) and type of content. However, it is acknowledged that sub-pixel estimation offers significant performance gains and modern coding standards all include the option to predict motion with fractional-pixel accuracy. MPEG-1 was the first international standard to use half-pixel estimation, closely followed by MPEG-2 in 1994. H.263 introduced more flexible block sizes and quarter-pixel estimation in 1996 and its improvement in performance over H.261 has been largely attributed to this innovation [5].

It is worth spending a few moments discussing the interpolation needed to support sub-pixel motion estimation. We could adopt a number of approaches:

1. Interpolate all of the current frame block and all of the search window and perform the entire motion estimation process at sub-pixel resolution.
2. Interpolate just the search window and proceed as in (1).
3. Perform integer-pixel search initially, then refine this locally using an interpolated search window and an interpolated current block.
4. Perform integer search initially, then refine this locally just interpolating the search window.

In practice method (4) is adopted, as it offers the lowest complexity with only a minor fall-off in performance (see Figure 9.7). This approach is illustrated in Figure 9.8, which shows the black square as the best full-pixel solution. The grid is then interpolated locally at half-pixel resolution and a refinement search is performed to obtain the

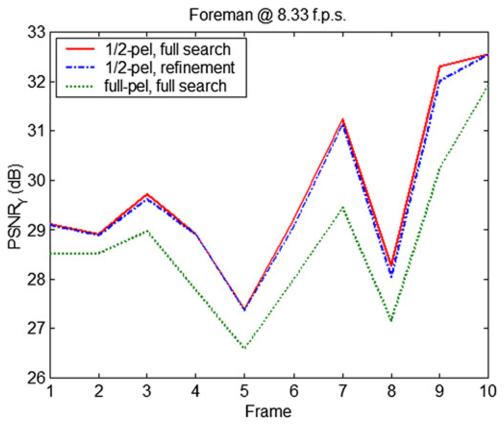


FIGURE 9.7

Sub-pixel search with local refinement.

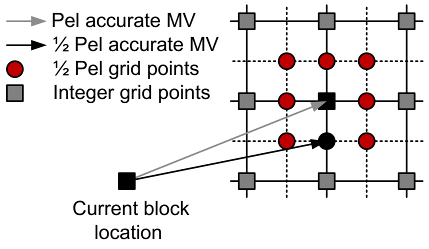


FIGURE 9.8

Motion estimation search grid with half-pixel refinement.

best half-pixel solution (black circle). The refinement search is normally performed exhaustively and the same matching criterion used as for the integer-pixel case.

9.3.2 Interpolation methods

Figure 9.9 illustrates the interpolation method used for half-pixel and quarter-pixel interpolation in H.264/AVC. In the case of half-pixel estimation, the search window is interpolated locally around the optimum integer-pixel result. If we assume that the pixels in the frame's search window are locally upsampled by a factor of 2 (see Chapter 6 for a more detailed discussion of upsampling), then the upsampled result in the x direction is given by:

$$s_u[x] = \begin{cases} s[x/2]; & x = 2m \\ 0; & \text{otherwise} \end{cases} \tag{9.5}$$

where m is an integer. We can then filter the result to interpolate the half-pixel locations. Normally the filter used is symmetrical and separable to reduce complexity. In the case of H.264/AVC [6], the filter used to create an interpolated half-pixel value, $g[x]$, is as follows:

$$g\left[x\right] = \sum_{i=-5}^{+5} h\left[i\right] s_u\left[x-i\right] \tag{9.6}$$

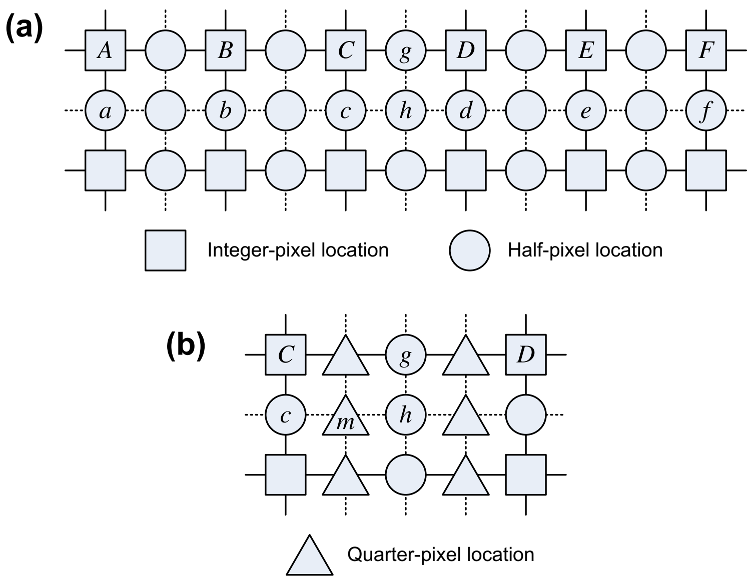


FIGURE 9.9

H.264 sub-pixel interpolation. (a) Half-pixel interpolation. (b) Quarter-pixel interpolation.

where the filter coefficients are:

$$h[i] = \{1, 0, -5, 0, 20, 0, 20, 0, -5, 0, 1\}/32 \tag{9.7}$$

In simple terms this means, with reference to the labeling in Figure 9.9, that:

$$g = \frac{(A - 5B + 20C + 20D - 5E + F)}{32} \tag{9.8}$$

and a similar approach is used vertically and on intermediate pixels such as h, based on pixels $\{a, b, c, d, e, f\}$.

In the case of quarter-pixel refinement, a simpler interpolation is employed; for example, to obtain the interpolated quarter-pixel value for location m in Figure 9.9, we use the following equation:

$$m = \frac{(c + h)}{2} \tag{9.9}$$

All interpolated results are rounded into the range $0 \cdots 255$, prior to the execution of the motion search phase.

It should be noted that the choice of interpolation filter is important as it can introduce bias (see for example the work of Bellers and de Haan [7]).

Example 9.3 (Sub-pixel motion estimation)

Implement the full search block matching motion estimation algorithm on the 6×6 search window $\mathbf{S}_{k-1}$, using the current frame template $\mathbf{S}_k$, given below. Use a simple two-tap interpolation filter to refine the integer-pixel result to half-pixel accuracy:

$$\mathbf{S} = \begin{bmatrix} 1 & 5 & 4 & 9 & 6 & 1 \\ 6 & 1 & 3 & 8 & 5 & 1 \\ 5 & 7 & 1 & 3 & 4 & 1 \\ 2 & 4 & 1 & 7 & 6 & 1 \\ 2 & 4 & 1 & 7 & 8 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}; \ \mathbf{M} = \begin{bmatrix} 3 & 9 \\ 1 & 4 \end{bmatrix}$$

Solution. SAD values for selected integer-pixel $[d_x, d_y]$ candidates are:

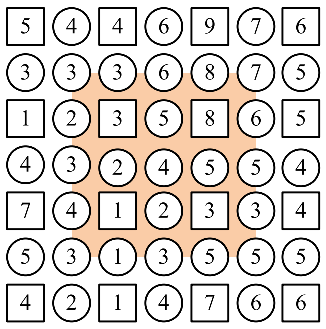
i	j	SAD(i,j)	i	j	SAD(i,j)	i	j	SAD(i,j)
−1	−1	17	0	1	7	−1	−2	8
−1	0	18	1	−1	11	−2	−2	14
−1	1	15	1	0	13	−2	−1	18
0	−1	2	1	1	17	−2	0	5
0	0	11	0	−2	7	2	−2	18

Clearly by inspection, all other candidate vectors give SAD values greater than 2. Hence the integer-pixel motion vector for this block is $\mathbf{d} = [0, -1]$.

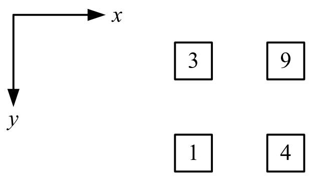
Next compute the half-pixel accurate motion vector. Use a simple two-tap filter to interpolate the values of the reference frame only. For example, in the horizontal direction:

$$s[x+0.5, y] = \frac{s[x,y] + s[x+1,y]}{2}$$

Once all values have been interpolated it is conventional to round the result prior to computing the sub-pixel SAD. We can use: $\lceil s[x+0.5] - 0.5 \rceil$ to round down values ending in 0.5. Using this approach, the interpolated half-pixel values in the vicinity of the integer-pixel result are given below:



Half-pixel interpolation



Current block pixels

and the sub-pixel SAD values are given in the following table:

i	j	SAD(i,j)	i	j	SAD(i,j)
0	−1	2	0.5	−1.5	>2
−0.5	−1	10	0	−0.5	>2
0.5	−1	7	−0.5	−0.5	>2
0	−1.5	>2	0.5	−0.5	>2
−0.5	−1.5	>2	–	–	–

Hence the half-pixel motion vector for this block is the same as before, i.e. $\mathbf{d} = [0, -1]$.

9.3.3 Performance

The benefits of fractional-pixel search are shown in Figures 9.10 and 9.11. Figure 9.10 shows the PSNR improvements obtained from using half- and quarter-pixel estimation for ten frames of the *Foreman* sequence (CIF at 25 fps). This is based on using the original frame as the reference in each case. It can be seen that up to 1.5 dB gain is achieved in this case. It can also be observed that, while quarter-pixel search does provide additional benefits, the gain is reduced. In most cases search at one-eighth pixel resolution will provide some small additional gain, but going beyond this provides diminishing returns.

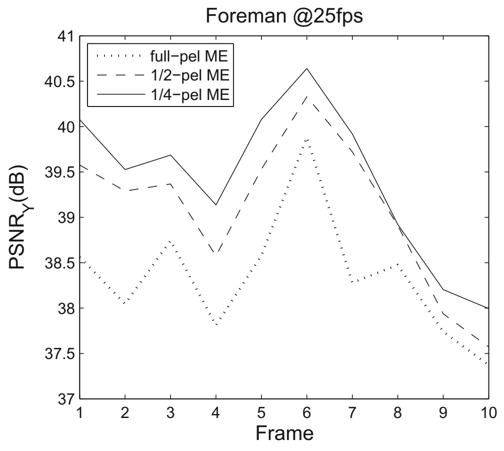


FIGURE 9.10

Comparison of full, half-, and quarter-pixel motion estimation.

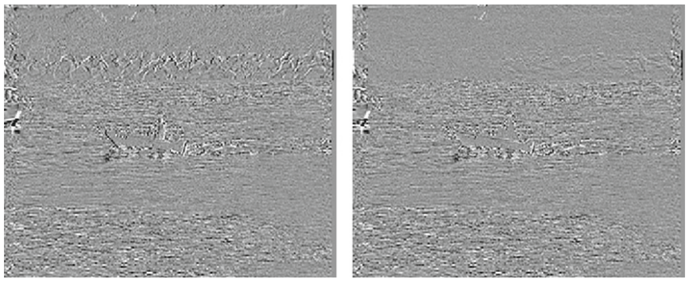


FIGURE 9.11

Influence of sub-pixel motion estimation on residual error. Left: integer-pixel ME. Right: half-pixel ME.

Figure 9.11 shows an example DFD signal for *Coastguard*, comparing integer- and half-pixel search methods. The reduction in residual energy is clear, especially in the region of the boat and the riverbank in the background. However, while there is some improvement in the region of the river, the dynamic textures of the water cause severe problems for the translational motion model. Techniques which model such textures using dynamic models will be discussed further in Chapter 13. It has been found that the use of quarter-pixel accurate motion prediction in H.264/AVC contributes between 10% and 20% saving in bit rate, compared with the integer-pixel case.

9.3.4 Interpolation-free methods

We have seen that quarter-pixel motion estimation can provide a significant contribution to the rate–distortion performance of a video codec. By using sub-pixel estimation, based on interpolated search values, the residual energy for each predicted macroblock can be significantly reduced, but only at the cost of increased computational complexity. This latter point is particularly relevant for codecs such as H.264/AVC and HEVC, where the cost of an exhaustive set of macroblock segmentations needs to be estimated for optimal mode selection.

To mitigate the cost of interpolation, novel schemes for sub-pixel motion estimation based solely on the whole-pixel SAD distribution have been proposed. The approach of Hill et al. [8] enables both half-pixel and quarter-pixel searches to be guided by an intelligent model-free estimation of the SAD surface. They use a method based on a 2-D parameterized parabolic interpolation between the estimated ambiguity samples, approximating the actual behavior in the presence of pure translations. They also include a fallback strategy that improves the rate–distortion performance to a level close to full interpolation. This approach is faster (in terms of frames per second) than a fully interpolated system by a factor of between 12% and 20%.

Hill and Bull [9] extended the method of Ref. [8] using a two-dimensional kernel approach. This reduces the number of quarter-pixel search positions by a factor of 30% giving an overall speed-up of approximately 10% compared to the EPZS quarter-pixel method.

9.4 Multiple reference frame motion estimation

9.4.1 Justification

In conventional motion estimation, only one reference frame is employed. However, it was observed by Wiegand et al. [10] that such approaches do not exploit long term statistical dependencies in video sequences and that *multiple reference frame motion estimation* (MRF-ME) can provide many benefits. Multiple reference frames were first standardized in H.263 and have been a feature in most codecs since that time [6]. Normally the frames employed as references are simply past decoded frames; however, they may also include warped versions to account for various camera motions.

Reference frames are assembled in a reference picture buffer at both the encoder and decoder, and synchrony of contents must be maintained at both ends. This is illustrated in Figure 9.12. This is simple when a sliding window update is employed but requires signaling when other, more intelligent updates are used.

9.4.2 Properties, complexity, and performance of MRF-ME

Properties

A good description of the properties of long term memory is provided by Al-Mualla et al. [12] and the reader is referred there for more details. To summarize (assuming a sliding window update method):

1. The distribution of long term memory spatial displacements is center-biased. This means that $\mathbf{d} = [0, 0, d_z]$ has the highest relative frequency of occurrence.
2. The distribution of long term memory temporal displacements is zero-biased. This means that the most recent reference frame is the most commonly adopted candidate. i.e. $\mathbf{d} = [0, 0, 0]$ is the most common vector.

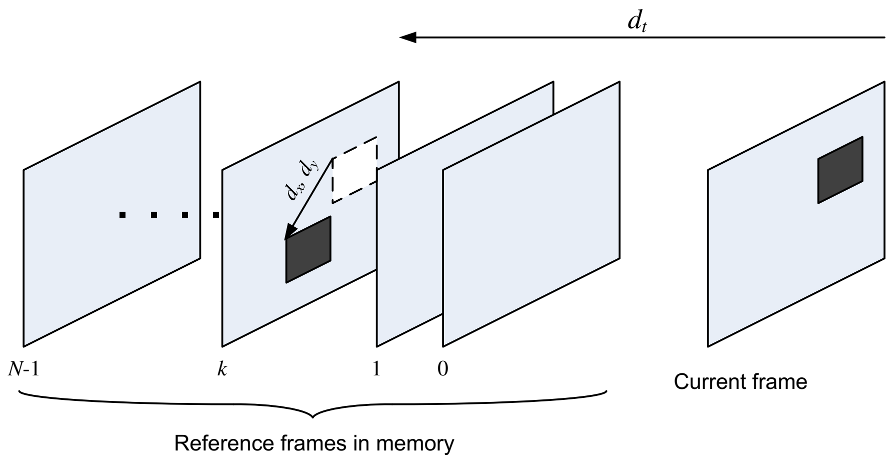


FIGURE 9.12

Multiple reference frame motion estimation.

3. The long term memory motion field is smooth and slowly varying. Hence search methods that work well for single-frame search can also offer solutions for the multiple-frame case.

Performance and complexity

MRF-ME exploits the long term statistical dependencies in an image sequence to provide coding gain. However, in order to achieve this, the motion estimation algorithm must not only search across spatial locations in the search grid, but also temporally across multiple candidate frames. With increases in processor power and the increasing integration and lowering cost of memory, this approach is now feasible. Nonetheless it can still present a major computational overhead and reduced complexity approaches are desirable. Figure 9.13 (left) illustrates the performance gains possible for the case of *Foreman* (QCIF @25 fps). The figure shows the benefits of long term memory up to 50 reference frames, indicating an increase in PSNR from 33 dB for one frame, up to 35.5 dB for 50 frames.

This improvement in coding performance is, however, offset by the additional complexity required to search for the optimum motion vector. This is shown in Figure 9.13 (right) which demonstrates, for the same sequence, that the number of locations searched per frame (for full search) increases from 80,000 for the case of one reference frame to over 4,000,000 for 50 reference frames. Such an increase in complexity, even with today's technology, could make MRF-ME intractable.

9.4.3 Reduced complexity MRF-ME

Several methods for improving the efficiency of MRF-ME without compromising its associated coding gain have been proposed. Su and Sun [11] exploit the smoothness

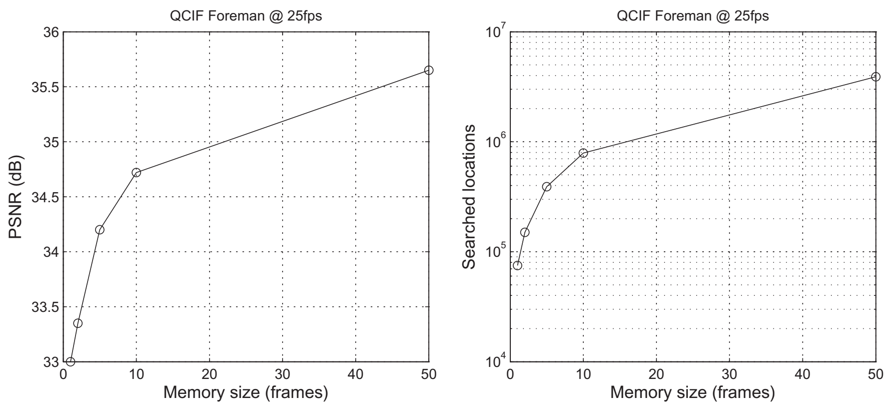


FIGURE 9.13

MRF-ME performance. Left: PSNR. Right: complexity.

Table 9.1 Performance of simplex-based (SMS) multiple reference frame motion estimation. Results from Ref. [13].

Algorithm:	FS	MR-FS	MR-SMS
PSNR (dB):	32.2	33.97	33.87
Search locations/frame	77,439	3,554,700	106,830

and correlations within the block motion field to provide similar coding performance to the H.264/AVC JM but at around 20% of the complexity. Al-Mualla et al. [13] showed that their simplex method can provide benefits for long term memory motion estimation. The simplex-based approach has a number of useful properties that make it attractive in such situations. For example, it is not based inherently on a unimodal error surface and it offers diversity in its candidate search locations. Table 9.1 presents coding performance and complexity results for this algorithm; this is based on 300 frames of *Foreman* (QCIF @25 fps), using an SAD distortion measure with a block size of 16 $\times$ 16 pixels and a maximum motion displacement of 15 pixels. The search was performed to full-pixel accuracy over 50 reference frames.

It can be seen that significant gains in performance are possible using this approach; a coding gain decrease of only 0.1 dB is accompanied by a speed-up of some 33 times.

9.5 Variable block sizes for motion estimation

9.5.1 Influence of block size

One of the main reasons for the performance gains of recent standards has been the introduction of variable block sizes, both for transforms and for motion estimation.

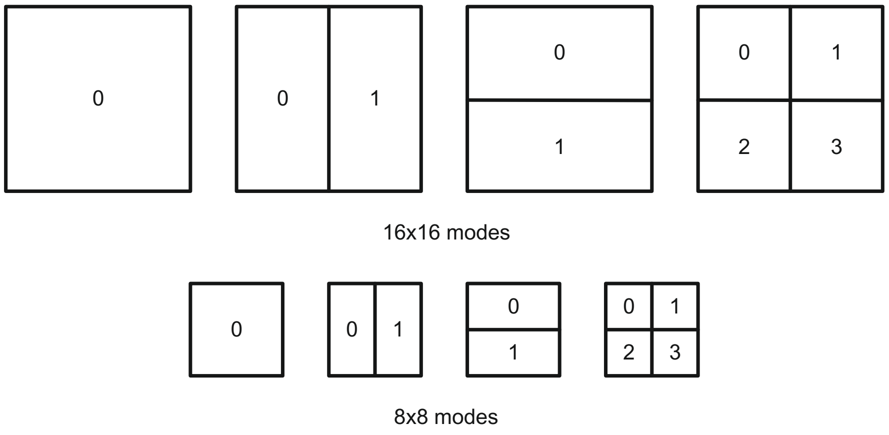


FIGURE 9.14

Variable block sizes supported by H.264/AVC. Top: 16×16 modes 1–4. Bottom: 8×8 modes 1–4.

This enables the shape and size of the block to be optimized in a rate–distortion sense, taking account of content type, spatial and temporal characteristics, and bit rate constraints. We saw in Chapter 8 that block size can influence both DFD accuracy and motion bit rate overhead. It thus seems perfectly sensible that block sizes should not be fixed, but instead should be allowed to adapt to content type within a rate–distortion framework.

9.5.2 Variable block sizes in practice

The block sizes supported by H.264/AVC are:

- **16 x 16 modes:** 16×16, 16×8, 8×16, 8×8.
- **8 x 8 modes:** 8×8, 8×4, 4×8, 4×4.

These are shown in Figure 9.14 and have been found to offer approximately 15% saving over the use of fixed block sizes. The distribution of block sizes in an actual coded frame is shown in Figure 9.15 for the *Foreman* sequence, produced using the Elecard Streameye analyzer [14].

9.6 Variable sized transforms

9.6.1 Integer transforms

Recent standards such as H.264/AVC [6] and HEVC [15] use block transforms to code the intra-frames or prediction residuals, but, rather than employ a single transform type, variable sizes are employed. For example, in H.264/AVC, the smallest block

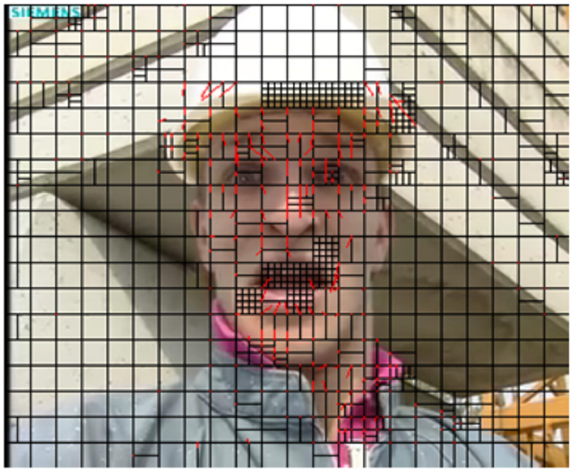


FIGURE 9.15

Block size distribution (Foreman sequence coded using H.264/AVC). Produced using Ref. [14].

size is 4×4 and a separable integer transform which is closely related to the 4×4 DCT is used. This integer transform was introduced by Malvar et al. [16] and offers benefits:

- The transform block size is matched to the basic motion estimation block, so it performs better in the presence of complex local motions and regions with random textures.
- Because of the improvements resulting from variable block sizes in HEVC and H.264/AVC, the residual signal after motion compensation has reduced spatial correlation. As discussed previously, this implies that the transform is less efficient and hence a 4×4 integer transform can often be as good as a larger transform in terms of decorrelation.
- It offers the benefits of reducing noise around edges.
- The inverse transform is matched to the forward transform with exact integer operations, hence encoder–decoder mismatches are eliminated.
- The integer transform is computationally efficient requiring a smaller processing wordlength, and can be implemented with just addition and shift operations.

In Chapter 5 we saw that the 1-D 4×4 DCT is given by:

$$\mathbf{A} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \sqrt{\frac{1}{2}}\cos\left(\frac{\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{3\pi}{8}\right) & -\sqrt{\frac{1}{2}}\cos\left(\frac{3\pi}{8}\right) & -\sqrt{\frac{1}{2}}\cos\left(\frac{\pi}{8}\right) \\ \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ \sqrt{\frac{1}{2}}\cos\left(\frac{3\pi}{8}\right) & -\sqrt{\frac{1}{2}}\cos\left(\frac{\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{\pi}{8}\right) & -\sqrt{\frac{1}{2}}\cos\left(\frac{3\pi}{8}\right) \end{bmatrix}$$

$$= \begin{bmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & -c \end{bmatrix} \quad \text{where:} \quad \begin{aligned} a &= \tfrac{1}{2} \\ b &= \sqrt{\tfrac{1}{2}}\cos\left(\tfrac{\pi}{8}\right) \\ c &= \sqrt{\tfrac{1}{2}}\cos\left(\tfrac{3\pi}{8}\right) \end{aligned} \tag{9.10}$$

Letting $d = \frac{c}{b}$, the integer version of the transform can be written as follows:

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & d & -d & -1 \\ 1 & -1 & -1 & 1 \\ d & -1 & 1 & -d \end{bmatrix} \otimes \begin{bmatrix} a & a & a & a \\ b & b & b & b \\ a & a & a & a \\ b & b & b & b \end{bmatrix}$$

where $\otimes$ represents a scalar multiplication. Since $d \approx 0.414$, the next step is to approximate d by a value of 0.5, to give:

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \otimes \begin{bmatrix} a & a & a & a \\ \frac{b}{2} & \frac{b}{2} & \frac{b}{2} & \frac{b}{2} \\ a & a & a & a \\ \frac{b}{2} & \frac{b}{2} & \frac{b}{2} & \frac{b}{2} \end{bmatrix} \tag{9.11}$$

Since for all row vectors, $\mathbf{a}_i\mathbf{a}_j^{\mathrm{T}} = 0$, the basis vectors remain orthogonal. To make the matrix orthonormal, we must ensure that $\|\mathbf{a}_i\| = 1$. Hence:

$$a = \frac{1}{2}; \quad b = \sqrt{\frac{2}{5}}; \quad d = \frac{1}{2}$$

Extending this separable transform to process 2-D signals gives:

$$\begin{aligned} \mathbf{C} &= \mathbf{ASA}^{\mathrm{T}} \\ &= \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} [\mathbf{S}] \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{bmatrix} \otimes \mathbf{E} \end{aligned} \tag{9.12}$$

where:

$$\mathbf{E} = \begin{bmatrix} a^2 & \frac{ab}{2} & a^2 & \frac{ab}{2} \\ \frac{ab}{2} & \frac{b^2}{4} & \frac{ab}{2} & \frac{b^2}{4} \\ a^2 & \frac{ab}{2} & a^2 & \frac{ab}{2} \\ \frac{ab}{2} & \frac{b^2}{4} & \frac{ab}{2} & \frac{b^2}{4} \end{bmatrix}$$

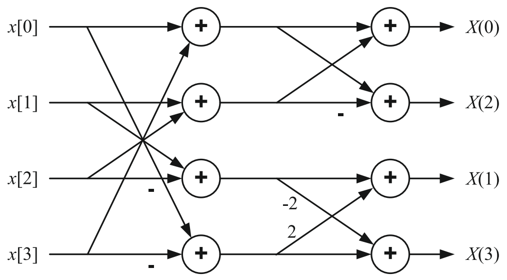
Thus we can perform the forward and inverse transform operations as: $\mathbf{S} = \mathbf{ACA}^{\mathrm{T}}$ with the scalar multiplication by $\mathbf{E}$ absorbed into the quantization process.

9.6.2 DC coefficient transforms

We saw earlier that intra-prediction for luma and chroma signals can provide significant benefits for coding non-textured regions. If a smoothly varying region extends across the whole coding block, then additional benefits can be obtained by performing a further transform on the DC coefficients from the 4×4 transforms. In H.264, a 2×2 transform is also applied to the DC coefficients of the four 4×4 blocks of each chroma component. This provides benefits because, for smoothly varying regions, the autocorrelation coefficient will approach unity and the reconstruction accuracy is proportional to the inverse of the size of the transform. For non-textured regions the reconstruction error for an 8×8 transform is therefore half of that for a 4×4 transform.

Example 9.4 (Reduced complexity integer transform)
Consider the forward integer 4×4 transform given in Eq. (9.11). Show that this can be implemented using only shift and add operations and compute its complexity.

Solution. The architecture for the fast integer transform is given below:



It can be observed that the transform requires only eight additions/subtractions and two 1-bit shifts.

9.7 In-loop deblocking operations

Blocking artifacts can arise in the hybrid video codec because of motion estimation and transform coding of the DFD signal. Because of this, deblocking filters have been used in various forms in standardized video codecs since H.261. They have been proven to improve both objective and subjective performance and have become progressively more sophisticated over time. Wiegand et al. [6] and List et al. [17] provide excellent overviews of the deblocking filter used in H.264/AVC; a summary is given below.

Deblocking filters perform best when they are integrated into the motion prediction loop, as this eliminates drift and enables benefits for all frame types. The idea of the deblocking operation is illustrated in Figure 9.16. The edge samples are filtered according to a set of criteria that relate to the prevailing quantization conditions coupled with estimates of whether the edge is real or induced by coding. The absolute differences of pixels near a block edge are first computed and then referenced against

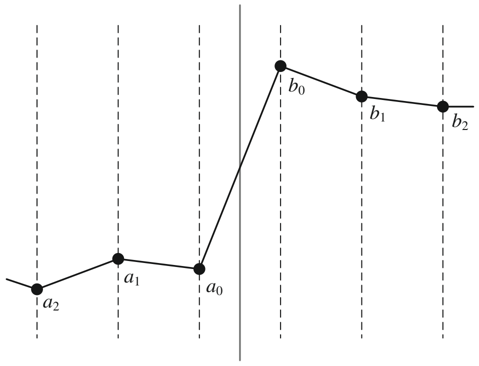


FIGURE 9.16
One-dimensional edge example.

Algorithm 5 Deblocking operation.

```
1. Set thresholds $\alpha(QP)$ and $\beta(QP)$;
2. Filter $a_0$ and $b_0$ iff: $|a_0 - b_0| < \alpha(QP)$ AND $|a_1 - a_0| < \beta(QP)$ AND
   $|b_1 - b_0| < \beta(QP)$;
3. Filter $a_1$ and $b_1$ iff: $|a_2 - a_0| < \beta(QP)$ OR $|b_2 b_0| < \beta(QP)$.
```

the likely size of any quantization-induced artifact. If the edge is larger than any that might have been induced by coding, then it is most likely to be an actual edge and should not be filtered. It has been reported [6] that the incorporation of the deblocking filter saves as much as 10% in bit rate for equivalent quality.

The conditions for filtering to be invoked depend on two thresholds $\alpha(QP)$ and $\beta(QP)$ where the second threshold is much smaller than the first. The deblocking process is described in Algorithm 5.

Example frames from *Foreman* with and without a deblocking filter applied are shown in Figure 9.17. The subjective benefits of the filtering operation can be clearly

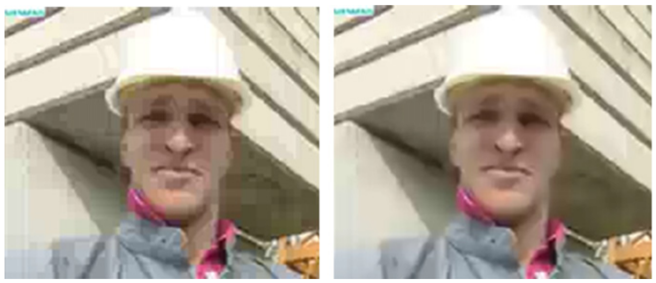


FIGURE 9.17
Illustration of the effect of a deblocking filter. Left: without deblocking. Right: with deblocking filter applied.

seen. What cannot be seen directly are the benefits of the smoothed and better correlated DFD signal which will enable more efficient transform coding and hence reduced bit rate.

9.8 Summary

This chapter has integrated the concepts from Chapters 4, 5, 7, and 8 to form what is referred to as a block-based hybrid coder. This architecture has been universally adopted through standardization processes and is the workhorse of all of our TV, storage and streaming applications. This basic architecture has benefited from several enhancements by way of advancing international standards. Some of the key enhancements have been described in this chapter and it is these that are responsible for providing the excellent performance offered by recent standards such as H.264/AVC and HEVC.

For interactive demonstrations of video codec operation, the reader is referred to the Matlab code associated with this text, which can be downloaded from www.bristol.ac.uk/vi-lab/demos.

References

[1] P. Strobach, Tree-structured scene adaptive coder, IEEE Transactions on Communications 38 (4) (1990) 477–486.
[2] Recommendation ITU-T H.264 | International Standard ISO/IEC 14496-10, ITU-T, 2013.
[3] S. Ericsson, Fixed and adaptive predictors for hybrid predictive/transform coding, IEEE Transactions on Communications 33 (12) (1985) 1291–1302.
[4] B. Girod, Motion-compensated prediction with fractional-pel accuracy, IEEE Transactions on Communications 41 (4) (1993) 604–612.
[5] B. Girod, E. Steinbach, N. Farber, Performance of the H.263 compression standard, Journal of VLSI Signal Processing 17 (1997) 101–111.
[6] T. Wiegand, G. Sullivan, G. Bjøntegaard, A. Luthra, Overview of the H.264/AVC video coding standard, IEEE Transactions on Circuits and Systems for Video Technology 13 (7) (2003) 560–576.
[7] E. Bellers, G. de Haan, Analysis of sub-pixel motion estimation, in: Proceedings of SPIE Visual Communications and Image Processing, 1999, pp. 1452–1463.
[8] P. Hill, T. Chiew, D. Bull, C. Canagarajah, Interpolation-free subpixel accuracy motion estimation, IEEE Transactions on Circuits and Systems for Video Technology 16 (12) (2006) 1519–1526.
[9] P. Hill, D. Bull, Sub-pixel motion estimation using kernel methods, Signal Processing: Image Communication 25 (4) (2010) 268–275.
[10] T. Wiegand, X. Zhang, B. Girod, Long term memory motion-compensated prediction, IEEE Transactions on Circuits and Systems for Video Technology 9 (1) (1999) 70–84.
[11] Y. Su, M.-T. Sun, Fast multiple reference frame motion estimation for H.264/AVC, IEEE Transactions on Circuits and Systems for Video Technology 16 (3) (2006) 447–452.
[12] M. Al-Mualla, C. Canagarajah, D. Bull, Video Coding for Mobile Communications, Academic Press, 2002.

[13] M. Al-Mualla, C. Canagarajah, D. Bull, Simplex minimization for single- and multiple-reference motion estimation, IEEE Transactions on Circuits and Systems for Video Technology 11 (12) (2001) 1209–1220.
[14] <http://www.elecard.com/en/products/professional/analysis/streameye.html>.
[15] G. Sullivan, J.-R. Ohm, W.J. Han, T. Wiegand, Overview of the high efficiency video coding (HEVC) standard, IEEE Transactions on Circuits and Systems for Video Technology 22 (12) (2012) 1649–1668.
[16] H. Malvar, A. Hallapuro, M. Karczewicz, L. Kerofsky, Low-complexity transform and quantization in H.264/AVC, IEEE Transactions on Circuits and Systems for Video Technology 13 (2003) 598–603.
[17] P.A. List, J. Lainema, G. Bjøntegaard, M. Karczewicz, Adaptive deblocking filter, IEEE Transactions on Circuits and Systems for Video Technology 13 (2003) 614–619.