CHAPTER

8 Coding Moving Pictures: Motion Prediction

CHAPTER OUTLINE

© 2014 Elsevier Ltd. All rights reserved.

In previous chapters we have studied various important aspects of image and video coding. We have seen how to create an appropriate input format and color space in Chapter 4, we saw how to decorrelate and quantize images using transforms such as the DCT in Chapter 5 and the DWT in Chapter 6, and in Chapter 7 we described how to represent the quantized output using various forms of entropy coding. This chapter completes "the picture" and addresses the final component in our basic video codec—motion prediction. Motion estimation is an important component in video coding as it enables temporal decorrelation. So rather than code very similar information repeatedly in each frame, we instead code the residual signal after motion prediction, along with the prediction parameters (the motion vectors).

This chapter firstly introduces the idea of temporal correlation and discusses how it can be exploited to provide additional coding gain. It then compares the merits of various motion models. Block matching, based on a translational motion model, is the approach adopted in all standardized video codecs to date and this is studied in detail with its performance characterized in terms of the effects of block size, search range, and motion vector coding. Alternative approaches to motion estimation, such as those based on phase correlation, are also covered. Although such techniques have rarely been adopted for conventional video compression, they are worthy of mention because of their use in video standards conversion.

One of the primary issues associated with motion estimation, even today, is its complexity—if not optimized, motion estimation can account for 60–70% of the processor loading. Methods of reducing motion estimation complexity have thus received significant attention in recent years and selected low complexity methods are compared in Section 8.4 in terms of their accuracy–complexity trade-offs. Finally, Section 8.5 describes how we can efficiently code motion information prior to transmission or storage.

8.1 Temporal correlation and exploiting temporal redundancy

8.1.1 Why motion estimation?

For still natural images, significant spatial redundancies exist and we have seen that these can be exploited by transform, filter bank or other coding methods that provide spatial decorrelation. The simplest approach to encoding a sequence of moving images is thus to apply an *intra-frame* (still image) coding method to each frame (cf. Motion

JPEG or Motion JPEG2000). This, as we will see later, does have some benefits, especially in terms of avoiding error propagation caused by the temporal prediction loop used in most video codecs. It can also be invoked in cases where a motion model fails to capture the prevailing motion in the scene. However, it generally results in limited compression performance.

We have already seen in Chapter 1, for real-time video transmission over low bandwidth channels, that there is often insufficient capacity to code each frame in a video sequence independently (25–30 fps is required to avoid flicker). The solution to this problem lies in taking advantage of the correlation that exists between temporally adjacent frames in a video sequence. This *inter-frame* redundancy can be exploited through motion prediction to further improve coding efficiency. Figure 8.1 demonstrates the effects of this approach, showing the first 53 frames of the *Foreman* CIF sequence, coded with a GOP length of 50 at approximately 500 kb/s. Frames 1 and 51 are coded as intra-frames (I-frames) and the remainder are coded as inter-frames (or predicted frames, also known as P-frames).

A number of issues however arise with this approach and these will be discussed further in the following sections:

- Motion estimation yields a residual signal with reduced correlation. Hence transform coding is less efficient on *displaced frame difference* (DFD) signals than on intra-frames.
- Motion can be a complex combination of translation, rotation, zoom, and shear. Hence model selection is key in balancing motion prediction accuracy with parameter search complexities.
- Lighting levels can change from frame to frame—modifying average luminance levels.
- Occluded objects can become uncovered and visible objects can become occluded, confounding motion models and leading to model failure.

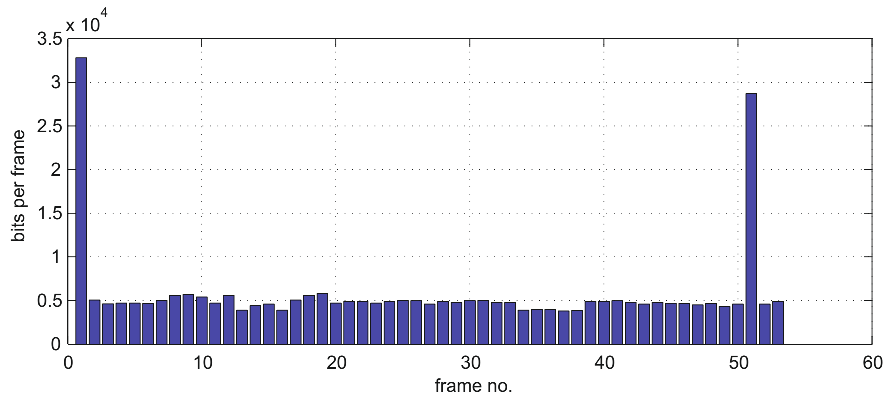


FIGURE 8.1

Variation in the number of bits per frame and frame coding mode.

- Motion estimation and compensation adds complexity to the codec.
- Motion information creates an overhead as it must be transmitted alongside the quantized transform coefficients.

Example 8.1 (Comparing intra- and inter-frame coding)
Consider a sequence with 512 × 512 spatial samples captured at 30 fps with 8 bits per color channel and a GOP length of 100 frames. Compare uncompressed, intra-compressed and inter-compressed bit rates for this sequence if (for a given quality) the average number of bits per pixel (including motion vector overheads) in intra-frame mode is 0.2 bpp per color channel, and 0.02 bpp in inter-frame mode.

Solution.

1. Uncompressed:

$$\text{Uncompressed bit rate} = 512 \times 512 \times 30 \times 24 = 189\ \text{Mbps}$$

2. Intra-frame compressed:

$$\text{Intra-compressed bit rate} = 512 \times 512 \times 30 \times 0.2 \times 3 = 4.7\ \text{Mbps}$$

3. Inter-frame compressed:

In this case 1 in every 100 frames is an intra-frame @ 0.2 bpp per color channel and 99 out of every 100 frames are inter-frames @ 0.02 bpp per color channel. Thus:

$$\begin{aligned}\text{Inter-compressed bit rate} &= 512 \times 512 \times 30 \times \left(\frac{0.2 \times 3 \times 1 + 0.02 \times 3 \times 99}{100}\right)\\ &= 514\ \text{kbps}\end{aligned}$$

If these sort of figures are achievable without compromising quality, then huge coding gains can be achieved by exploiting temporal redundancy. The goal is thus to exploit both temporal and spatial redundancy when coding the video sequence.

8.1.2 Projected motion and apparent motion

While, in the real world, objects move in three spatial dimensions, we are primarily interested here in the projection of their motion onto a 2-D image plane as captured by a camera. So, when we use the term "motion estimation" in the context of video compression, this normally refers to the *projected motion* of the object. While it could be argued that tracking an object in 3-D might provide a more accurate description of this projected motion, the models and methods associated with doing this are too complex to be of major interest.

In video compression, 2-D motion is conventionally estimated by observing the spatio-temporal variations in signal intensity between frames. This approach measures

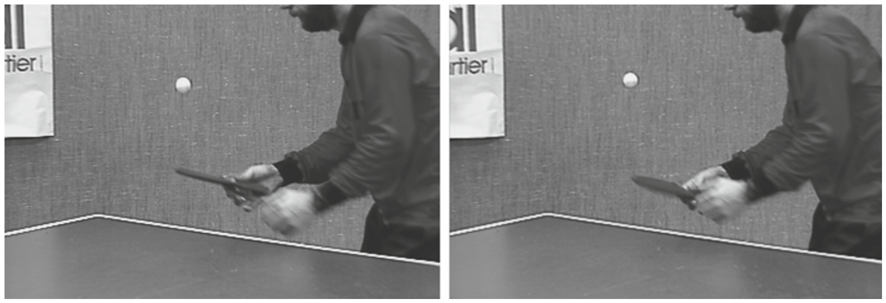


FIGURE 8.2

Temporal correlation between two adjacent video frames.

apparent motion rather than true projected motion and can lead to ambiguities. Consider, for example, a disk of uniform luminance rotating about its center. In this case the projected motion is clearly non-zero, whereas the apparent motion is zero. In contrast, an object that is static in the image plane, but experiencing time-varying illumination changes, will have zero projected motion but, because of the intensity changes between frames, will exhibit some apparent motion. The use of apparent motion can be justified since the aim of motion estimation in video compression is not to assess true motion, but instead to minimize the number of bits required to code the prediction residual.

We will see below that there are many possible motion models that could be adopted for motion estimation. However, the most commonly used is the simple 2-D translational model, where the displacement motion vector, $\mathbf{d} = \left[d_x, d_y\right]^T$. This is not normally computed for a whole frame, but instead for a number of smaller blocks—typically of size 16×16 pixels. The set of all vectors collectively represents the *motion field* for the given frame.[1]

8.1.3 Understanding temporal correlation

We showed in Chapter 3 how natural video data is correlated in time, with the temporal autocorrelation function typically remaining high across a wide range of lag values. This property can be exploited to improve coding efficiency by compensating for the movement of objects in a scene between adjacent frames via motion estimation and compensation (MEC). Take, for example, the pair of temporally adjacent frames from a typical 25 fps video sequence (*Table*) shown in Figure 8.2. Although this sequence has relatively high activity—the camera pans from right to left and there are clear movements of the player's hands, bat and ball—the two frames appear visually very similar. We will consider, in the following sections, how best to model this motion in order to improve coding gain.

[1]Formally for apparent motion, the displacement field is referred to as the correspondence field.

8.1.4 How to form the prediction

Our aim is to estimate the pixel values in the frame currently being encoded (the *current frame*), based on the values in one or more other frames (*reference frames*) in the sequence. These will often be temporally adjacent to the current frame but need not be.

There are two primary ways to form the prediction:

- **Frame Differencing:** This method does not employ any motion model (although arguably it could be referred to as a zero order prediction). The reference frame is, without any modification, directly used as the prediction for the current frame. The prediction error is thus the difference between the pixel values in the current frame and those co-located in the reference frame and is known as the *frame difference* (FD).
- **Motion-Compensated Prediction:** In this case a motion model is assumed and motion estimation (ME) is used to estimate the motion that occurs between the reference frame and the current frame. Once the motion is estimated, a process known as motion compensation (MC) is invoked to use the motion information from ME to modify the contents of the reference frame, according to the motion model, in order to produce a prediction of the current frame. The prediction is called a *motion-compensated prediction* (MCP) or a *displaced frame* (DF). The prediction error is known as the *displaced frame difference* (DFD) signal.

Figure 8.3 compares the motion residuals from both of these approaches for those frames from *Table* shown in Figure 8.2. It can be seen that the residual energy is significantly reduced for the case of motion estimation. Figure 8.4 reinforces this point, showing how the pdf of pixel values is modified for FD and DFD frames, compared to an original frame from the *Football* sequence.

Three main classes of prediction are used in video compression:

- **Forward Prediction:** The reference frame occurs temporally before the current frame.
- **Backward Prediction:** The reference frame occurs temporally after the current frame.

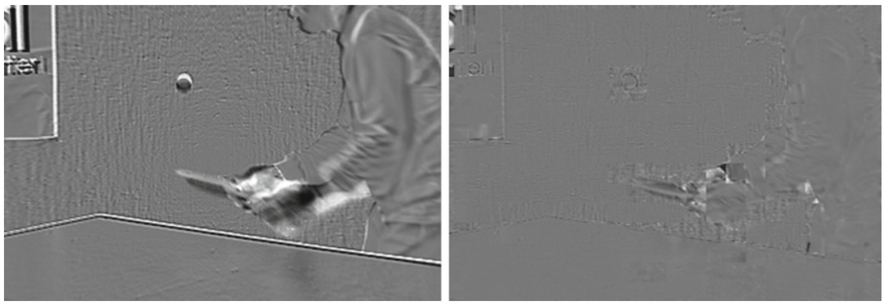


FIGURE 8.3

Frame differencing (left) vs motion-compensated prediction (right).

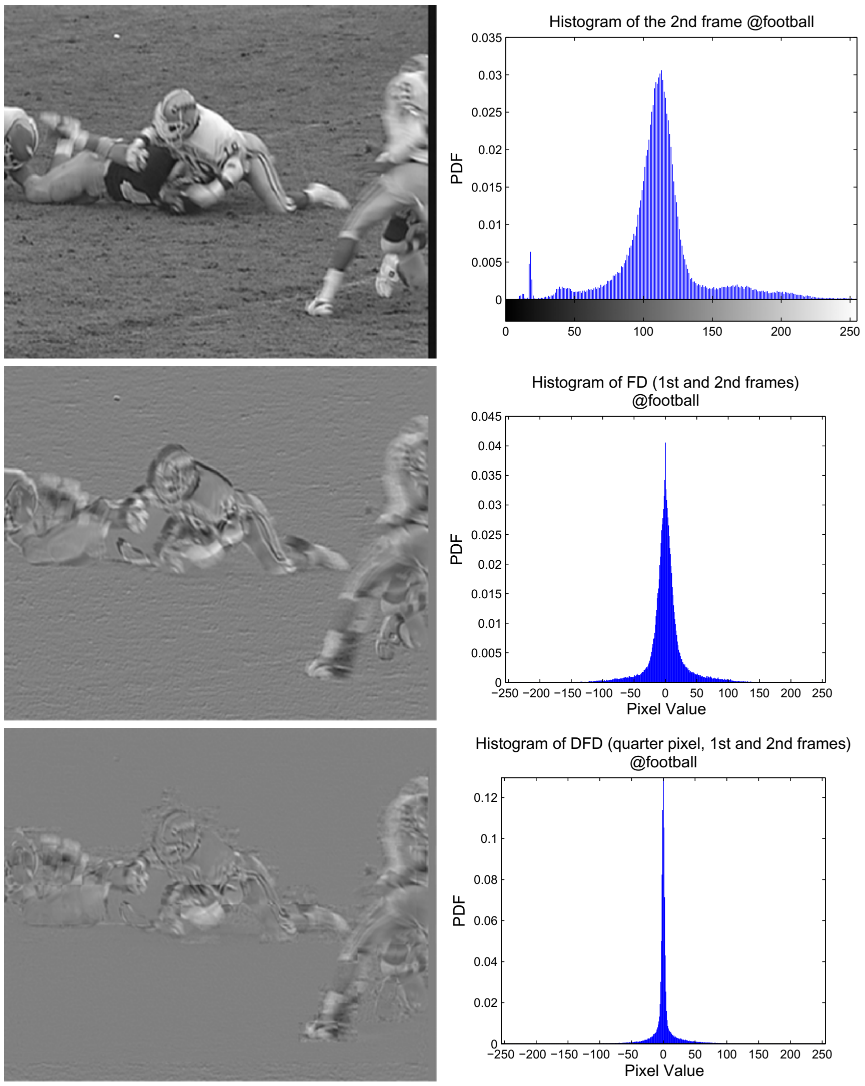


FIGURE 8.4
Probability distributions for an original frame and FD and DFD frames from the *Football* sequence.

- **Bidirectional Prediction:** Two (or more) reference frames (forward and backward) are employed and the candidate predictions are combined in some way to form the final prediction.

In *forward prediction*, assuming that the motion model used is perfect and based on translation only, the pixel value at location $\mathbf{p} = [x, y]^T$ in the current frame at time k

is related to a pixel location in the previous frame at time $k-1$ by:[2]

$$S_k[\mathbf{p}] = S_{k-1}[\mathbf{p} - \mathbf{d}] \tag{8.1}$$

where $\mathbf{d}$ is the motion vector corresponding to location $\mathbf{p}$ in the current frame. A similar expression can be generated for *backward prediction*. More sophisticated motion models are considered in the next section.

It should be noted that the motion estimation problem is ill-posed and there is not necessarily a unique solution—the number of independent variables exceeds the number of equations. In fact a solution does not necessarily exist at all, for example in the case of uncovered or occluded pixels. The estimate is also sensitive to the presence of noise.

8.1.5 Approaches to motion estimation

The key issues in motion estimation can be summarized as follows:

1. **The motion model:** This can be an accurate, but more complex motion model such as *affine* or *perspective*, or a simple but sufficiently effective model such as *translational*. In practice the translational block matching approach is adopted in most situations. Once the motion model is selected, we must decide how it will be implemented—for example, in the spatial or frequency domain.
2. **The matching criterion:** Once the model is selected, we must measure how good the match is for any candidate parameter set. This can be achieved with techniques such as normalized cross correlation (NCCF), sum squared differences (SSD), or sum of absolute differences (SAD).
3. **The region of support:** This is the group of pixels to which the motion model is applied and can range from a whole frame to a single pixel. The region can be arbitrarily shaped in two or three dimensions so as to best reflect the actual motion characteristics of a scene, but is normally based on rectangular blocks. In simple codecs these are typically 16×16 but variable block sizes have become commonplace.
4. **The search range:** This is the window of pixels in one or more reference frames within which the region of support is compared using the matching criterion, in order to estimate the motion and parameterize the model. The search range will depend on the spatio-temporal resolution of the format used and the activity level of the content coded. Normally a rectangular region is employed with a maximum offset between 7 and 63 pixels.
5. **The search strategy:** Once all of the above choices are made, we need to execute a means of efficiently finding the model parameter that best suit the motion. Because of the complexity associated with performing an exhaustive search, fast methods are commonly employed. In the case of translational block matching these include the *N-Step Search* (NSS), the *Diamond Search* (DS), and the *Hexagon-based Search Method* (HEXBS).

[2]Without loss of generality we use the adjacent frame here.

In all situations, there will exist a trade-off between prediction accuracy and coding overhead and appropriate choices have to be made based on processing power available, video format, and content type. The following sections explore these issues in more detail.

8.2 Motion models and motion estimation

8.2.1 Problem formulation

Apparent motion can be attributed to three main causes:

1. Global motion—e.g. camera motion (pan, tilt, zoom, and translation).
2. Local motion—of objects in the scene.
3. Illumination variations—due to changes in lighting, shadows, or noise.

The models and methods described in this chapter address the combined motion due to all three of these reasons. Other methods exist whereby global motion is extracted initially followed by local refinement (e.g. Ref. [1]) but these are not considered further here.

8.2.2 Affine and high order models

The apparent motion in an image or image region can be complex and experience translation, rotation, shear, and dilation as well as suffering from illumination changes between frames and occlusion by other objects (as illustrated in Figure 8.5). In order to represent such motion characteristics, it is necessary to use an appropriate motion model. Examples of such models are the Affine, Bilinear, and Perspective models as described by equations (8.2), (8.3), and (8.4) respectively.

Affine:

$$\begin{aligned} u &= g_x(x, y) = a_1x + a_2y + a_3 \\ v &= g_y(x, y) = a_4x + a_5y + a_6 \end{aligned} \tag{8.2}$$

Bilinear:

$$\begin{aligned} u &= g_x(x, y) = a_1xy + a_2x + a_3y + a_4 \\ v &= g_y(x, y) = a_5xy + a_6x + a_7y + a_8 \end{aligned} \tag{8.3}$$

Perspective:

$$\begin{aligned} u &= g_x(x, y) = \frac{a_1x + a_2y + a_3}{a_7x + a_8y + 1} \\ v &= g_y(x, y) = \frac{a_4x + a_5y + a_6}{a_7x + a_8y + 1} \end{aligned} \tag{8.4}$$

where location (x, y) is mapped to (u, v) using appropriate warping parameters $a_1 \cdots a_8$. An excellent review of image warping transforms is provided by Glasbey and Mardia in Ref. [2].

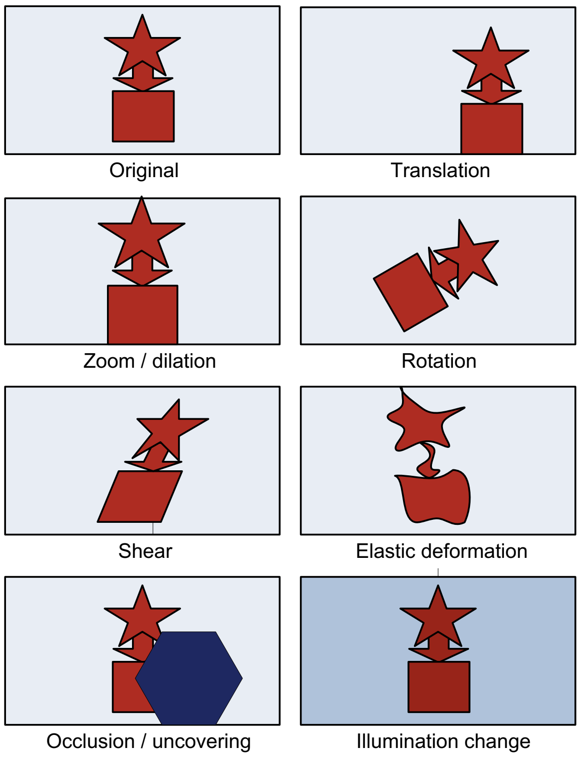


FIGURE 8.5

Motion-based image deformations.

Node-based warping

The use of higher order models has been widely reported in the research literature, but none have been adopted for standardization. There has however been recent renewed interest through the work of Zhang and Bull [3] and Vigars et al. [4] and several others. Furthermore, some authors propose the practice of using affine or eight-parameter models to model camera motion, while using a translational model for local motion based on small blocks.

In most cases, the warping parameters are obtained through the use of a 2-D deformable mesh formed from an interconnected set of node points, as shown in Figure 8.6. A method such as *block matching motion estimation* (BMME) is used to estimate the motion of each node through comparisons between a current frame and one or more reference frames, and this in turn is used to find the parameters $\{a_i\}$ of the motion model. The model is then used to transform each pixel in the current block region. Meshes can be continuous or discrete, fixed or adaptive and can adopt a range of approaches to node tracking. A good overview and comparison of these variations is provided in Ref. [5].

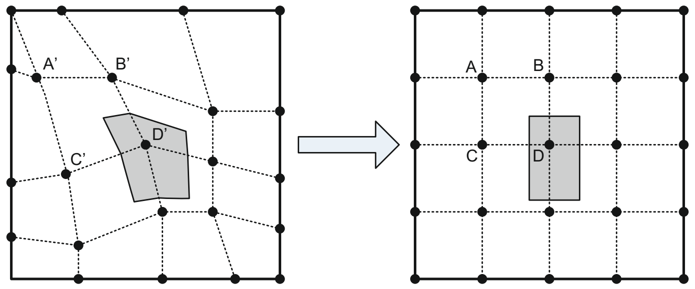


FIGURE 8.6

Mesh-based warping.

Once the nodes of the mesh have been tracked, the motion parameters can be obtained through solving a set of simultaneous equations, such as those shown in equation (8.5) for the bilinear model. Subsequently the transformation can be applied to each pixel in the patch using the appropriate model equation, in this case equation (8.3).

$$\begin{bmatrix} u_A & u_B & u_C & u_D \\ v_A & v_B & v_C & v_D \end{bmatrix} = \begin{bmatrix} a_1 & a_2 & a_3 & a_4 \\ a_5 & a_6 & a_7 & a_8 \end{bmatrix} \begin{bmatrix} x_A y_A & x_B y_B & x_C y_C & x_D y_D \\ x_A & x_B & x_C & x_D \\ y_A & y_B & y_C & y_D \\ 1 & 1 & 1 & 1 \end{bmatrix} \tag{8.5}$$

8.2.3 Translation-only models

The translational model is widely used in practice because it is simple to parameterize, it fits well into a regular block-based coding architecture and, importantly, it provides a good fit to the apparent motion in most scenarios. The latter point is worth explaining since it is obvious that true motion, projected motion, and apparent motion are not usually purely translational.

Recall that our primary purpose is to optimize rate–distortion performance, and this can be achieved with a simple motion model with low coding overhead (i.e. few parameters) and which delivers a low energy prediction residual. The translational model succeeds in this respect since, at typical video frame rates (25–50 fps) and spatial resolutions, most motion is approximately translational. This can be observed in Figures 8.2 and 8.3, which show two temporally adjacent frames together with their residual image after block-based translational motion estimation. While obvious small areas exist where the model fails, in most of the frame it works exceptionally well, providing a very low residual signal. Some further examples, where this assumption breaks down, are shown later.

The translational model is a subset of the perspective model, and is given by equations (8.6):

$$\begin{aligned} u &= g_x(x, y) = x + a_1 = x + d_x \\ v &= g_y(x, y) = y + a_2 = y + d_y \end{aligned} \tag{8.6}$$

Table 8.1 Comparison of warping and translational models for three test sequences [5].

PSNR (dB)	Akiyo	Foreman	Table
BMA	39.88	27.81	29.06
WBA	41.45	29.09	29.22
BMA-HO	41.77	29.51	29.87

It should also be noted that models embracing translation and rotation or translation and dilation are also possible. For example, the latter is given by:

$$\begin{aligned} u &= g_x(x, y) = cx + a_1 \\ v &= g_y(x, y) = cy + a_2 \end{aligned} \tag{8.7}$$

Al-Mualla et al. [5] compared warping methods with translational block matching, both based on 16×16 blocks, for low bit rate coding applications. The results are summarized in Table 8.1. Three algorithms were compared: BMA—a basic integer-pixel block matching algorithm, WBA—a warping-based algorithm based on 16×16 blocks with node warping using an eight-parameter model, and BMA-HO—BMA enhanced with half-pixel accuracy and overlapped block motion estimation. It can be seen that the warping approach does offer gains in prediction quality over the basic BMME algorithm but that this can be offset through enhancements to BMME such as sub-pixel estimation and the use of overlapped blocks. In terms of complexity (processor execution time), the BMA-HO was found to be 50% more complex than BMA, while WBA was found to be 50 times slower than BMA.

As mentioned above, the translational motion model is very widely used in practical video compression systems. We will return to it and describe it fully in Section 8.3.

8.2.4 Pixel-recursive methods

Pixel-recursive motion estimation methods were first introduced by Netravali and Robbins [6] and are based on the idea of iteratively minimizing the prediction error using a gradient descent approach. Given a displaced frame difference signal for a given frame (or block):

$$\mathrm{DFD}[\mathbf{p}, \mathbf{d}] = S_k[\mathbf{p}] - S_{k-1}[\mathbf{p} - \mathbf{d}] \tag{8.8}$$

the estimate for the motion vector at each location is then formed iteratively using a steepest descent approach, hence:

$$\hat{\mathbf{d}}^{i+1} = \hat{\mathbf{d}}^{i} - \varepsilon \mathrm{DFD}\left[\mathbf{p}, \hat{\mathbf{d}}^{i}\right] \nabla S_{k-1}\left[\mathbf{p} - \hat{\mathbf{d}}^{i}\right] \tag{8.9}$$

It has been observed that the convergence characteristics of pixel-recursive methods can be highly dependent both on the content (i.e. DFD characteristics) and on the step-size parameter ε. They struggle with smooth intensity regions and large displacements; they also inherently produce dense motion fields and this can lead to a high motion overhead unless parameter prediction methods are used. For these reasons, pixel-recursive methods have not been widely adopted in practice.

8.2.5 Frequency domain motion estimation using phase correlation (PhC)

Principles

The basic idea of phase correlation (PhC) was introduced by Kuglin and Hines [7] and refined by Thomas for broadcast applications [8]; it is illustrated in Figure 8.7. This method employs the 2-D Fourier transform to estimate the correlation between blocks in adjacent frames. Peaks in the correlation surface correspond to the cross correlation lags which exhibit the highest correlation between the current and reference blocks. One advantage of this approach is that the surface may possess multiple peaks, corresponding to multiple candidate motions. Some form of peak detection would then be required to establish the dominant motion or motion vector. Advantages of the PhC approach include:

- It can detect multiple motions in each block, facilitating object-based scene decomposition.
- Block size reduction and/or optimum vector selection for each pixel is possible.
- It can easily cope with large displacements.
- It offers robustness to noise and illumination variations as the Fourier phase is insensitive to mean shifts.
- It facilitates sub-pixel resolution through increasing the length of the DFT.

Assuming perfect translational block motion, for a block or picture, **S**, each location is related as follows:

$$S_k[x, y] = S_{k-1}[x + d_x, y + d_y] \tag{8.10}$$

where x and y represent the center pixel locations in block **S** of frame k, and d_x and d_y are the offset parameters which globally translate the block in frame k to align it with that in frame $k - 1$. From the 2-D DFT:

$$\mathcal{S}_k(f_x, f_y) = \mathcal{S}_{k-1}\left(f_x, f_y\right) \mathrm{e}^{j2\pi(d_x f_x + d_y f_y)} \tag{8.11}$$

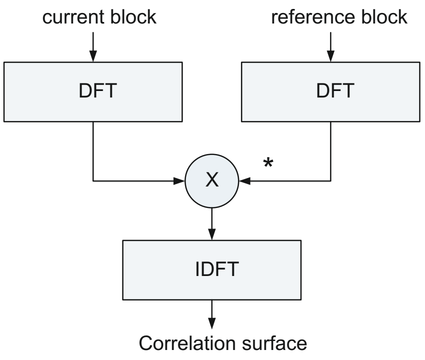


FIGURE 8.7

Phase correlation-based motion estimation.

In the case of translational motion, the difference of the 2-D Fourier phases in the respective blocks defines a plane in the spatial frequency variables f_x, f_y. The motion vector can be estimated from the orientation of this plane, thus:

$$\arg(\mathcal{S}_k(f_x, f_y)) - \arg(\mathcal{S}_{k-1}(f_x, f_y)) = 2\pi(d_x f_x + d_y f_y) \tag{8.12}$$

However, this requires phase unwrapping and it is not easy to identify multiple motions. Instead, the phase correlation method is used to estimate the relative shift between two image blocks by means of a normalized cross correlation function in the 2-D Fourier domain. In this case the correlation between frames k and $k-1$ is defined by:

$$C_{k,k-1}[x, y] = S_k[x, y] \star S_{k-1}[-x, -y] \tag{8.13}$$

Taking the Fourier transform of both sides, we obtain a complex cross-power spectrum:

$$\mathcal{C}_{k,k-1}[f_x, f_y] = \mathcal{S}_k(f_x, f_y)\mathcal{S}^*_{k-1}(f_x, f_y) \tag{8.14}$$

and normalizing this by its magnitude gives the phase of the cross-power spectrum:

$$\angle(\mathcal{C}_{k,k-1}[f_x, f_y]) = \frac{\mathcal{S}_k(f_x, f_y)\mathcal{S}^*_{k-1}(f_x, f_y)}{|\mathcal{S}_k(f_x, f_y)\mathcal{S}^*_{k-1}(f_x, f_y)|} \tag{8.15}$$

Assuming ideal translational motion by an amount $[d_x, d_y]$ we can obtain from equation (8.15):

$$\angle(\mathcal{C}_{k,k-1}[f_x, f_y]) = \mathrm{e}^{2\pi(d_x f_x + d_y f_y)} \tag{8.16}$$

and taking the inverse Fourier transform of equation (8.16) yields the phase correlation function:

$$C_{k,k-1}[x, y] = \delta[x - d_x, y - d_y] \tag{8.17}$$

i.e. the phase correlation function yields an impulse whose location represents the displacement vector.

Applications and performance

Phase correlation is the preferred method for TV standards conversion (e.g. frame rate conversion between US and EU frame rates), primarily due the fact that it produces a much smoother motion field with lower entropy than BMME, significantly reducing motion artifacts. Due to the use of the FFT, PhC can be computationally efficient. Phase correlation works well in the presence of large scale translational motion and it can handle multiple motions. However, the use of variable block size BM is now considered a more flexible approach and is favored for conventional motion estimation in modern video coding standards. An example of PhC is given in Figure 8.8.

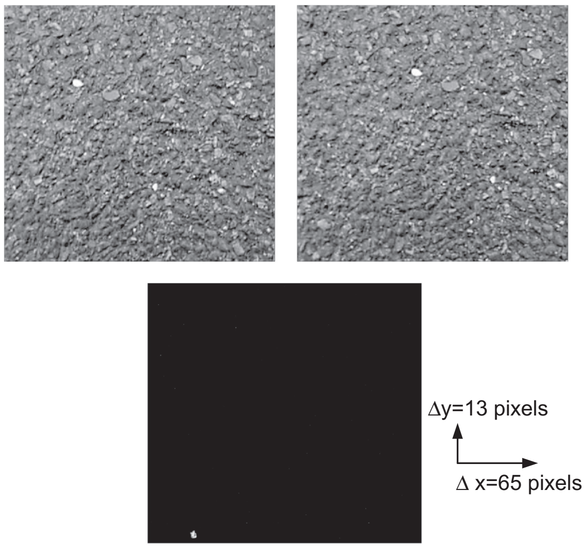


FIGURE 8.8

Example of phase correlation-based motion estimation. Top: input frames of a tarmac road surface taken with an approximately translating camera. Bottom: phase correlation surface.

8.3 Block matching motion estimation (BMME)

8.3.1 Translational block matching

Motion vector orientation

Figure 8.9 illustrates the convention we will adopt for motion vector orientation. The motion vector represents the displacement of the closest matching block in the reference frame with respect to the current block location (reversing the polarity of true motion). We will assume the [0,0] vector corresponds to the center of the search grid and offsets in x and y are positive right and down respectively.

Region of support—size of the search window

Intuitively, we would wish to search the whole of the reference picture for the best match to the current block. However, this is both impractical and unnecessary in most cases. Consider a 1080p25 sequence with an object traversing the screen in 1 s; with 1920 horizontal pixels, this means that the object moves horizontally by $1920/25 = 77$ pixels per frame. One could thus argue that such motion would dictate a search window of ± 77 pixels in order to predict it. In practice, a number of other factors come into play. Firstly, an object traversing a screen in 1 s is very fast and would rarely occur (not in directed content at least); secondly, if it did happen, there would be significant motion blur so the object could never be faithfully depicted anyway; and

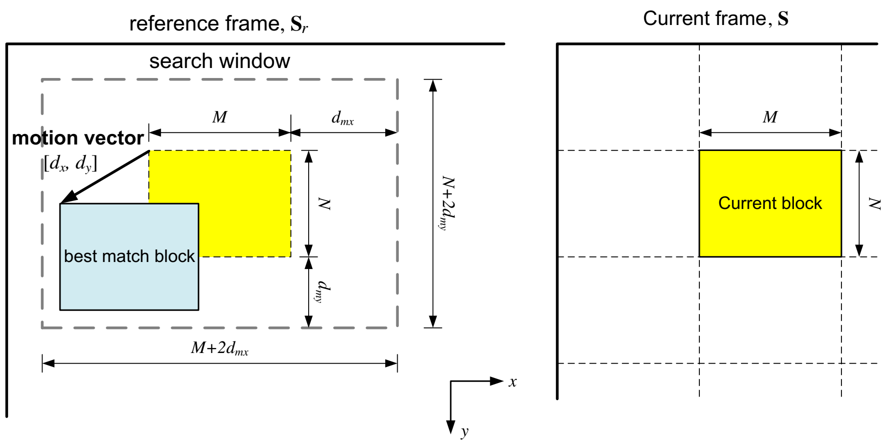


FIGURE 8.9

Translational block matching motion estimation (BMME).

thirdly, motion masking effects will exist which further diminish the need for faithful reproduction. As such, a compromise window is normally employed where the region of support caters for most motion scenarios while limiting search complexity.

The search grid is normally restricted, as shown in Figure 8.9, to a $[\pm d_{mx}, \pm d_{my}]$ window and, in most cases, $d_{mx} = d_{my} = d_m$. The value of d_m depends on image resolution and activity; for example, in broadcast applications, d_m typically ranges from 15 for low motion to 63 for high (sports) motion.

8.3.2 Matching criteria

The block distortion measure (BDM)

For block matching motion estimation, uniform motion and a translational motion model are normally assumed. We need to select the best motion vector for the current block and this will correspond to the lowest residual distortion. This is assessed with some form of block distortion measure (BDM), formulated as follows:

$$\text{BDM}(i, j) = \sum_{x=0}^{N-1} \sum_{y=0}^{M-1} g(s(x, y) - s_r(x + i, y + j)) \tag{8.18}$$

where $g(\cdot)$ is the function used to compute the BDM, i and j are the offsets for the current motion vector candidate, and x, y are the local coordinates within the current block.

We choose the motion vector that minimizes the BDM for the current block:

$$\mathbf{d} = [d_x, d_y]^{\mathrm{T}} = \arg\left(\min_{\forall(i,j)} (\text{BDM}(i, j))\right) \tag{8.19}$$

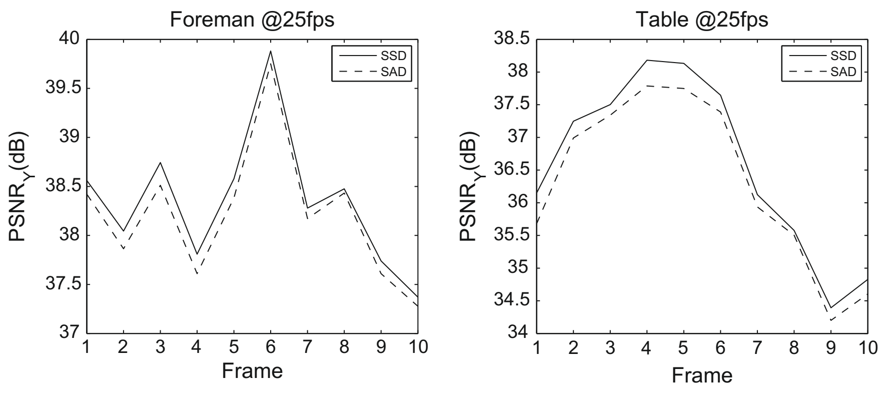


FIGURE 8.10

Effect of matching function: SAD vs SSD.

Absolute difference vs squared difference measures

There are many possible block distortion functions that could be used to assess the error residual after motion estimation. These include normalized cross correlation (NCCF), mean squared error (or sum of squared differences (SSD)), and sum of absolute differences (SAD). The choice of BDM function is key as it influences both the search complexity and the prediction accuracy.

The two most common functions are SAD (equation (8.20)) and SSD (equation (8.21)).[3]

$$\mathrm{SAD}(i,j) = \sum_{x=0}^{N-1}\sum_{y=0}^{M-1} |(s(x,y) - s_r(x+i, y+j))| \tag{8.20}$$

$$\mathrm{SSD}(i,j) = \sum_{x=0}^{N-1}\sum_{y=0}^{M-1} (s(x,y) - s_r(x+i, y+j))^2 \tag{8.21}$$

Figure 8.10 illustrates the difference in prediction accuracy between these two functions for two different sequences. It can be observed that, although the SSD metric consistently outperforms SAD in terms of quality, there is actually very little difference between the two methods. Furthermore, the SAD metric requires only $3NM$ basic operations (Add, Sub, Abs) per block to compute whereas the SSD requires NM multiplications plus $NM - 1$ additions. For this reason, the SAD function is frequently chosen over SSD.

[3]*Note:* SAD is normally used in preference to mean absolute difference (MAD).

8.3.3 Full search algorithm

The simplest BMME algorithm is based on *full search* or *exhaustive search*. In this case the global minimum of the error surface is found by exhaustively checking all possible $(2d_m + 1)^2$ motion vector candidates within the search grid.

8.3.4 Properties of block motion fields and error surfaces

The block motion field

For natural scenes, the block motion field is generally smooth and slowly varying. This is because there is normally high correlation between the motion vectors of neighboring blocks. Figure 8.11 illustrates this for the example of the *Foreman* sequence captured at 25 fps.

The effect of block size

Figure 8.12 shows the effect of different block sizes (8×8 vs 16×16) in terms of prediction quality and motion overhead. It can be seen that smaller block sizes normally result in better prediction accuracy, but only at the expense of significantly increased bit rates. The improved prediction accuracy is due to the fact that smaller blocks can deal better with multiple motions, complex motions (i.e. provide a better fit to the model), and the different properties of textured and non-textured regions. Because of the conflict between rate and distortion, many modern compression standards employ variable block sizes for motion estimation alongside rate–distortion optimization (RDO) techniques that select the block size that optimizes a rate–distortion trade-off. This is discussed further in Chapters 9 and 10.

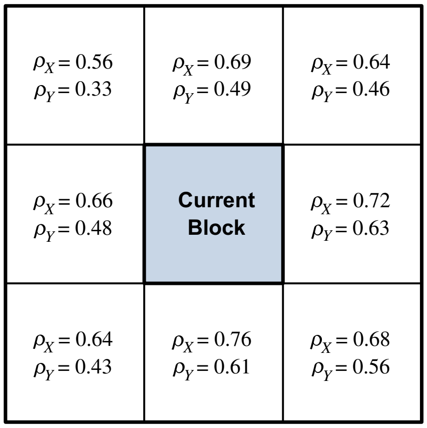


FIGURE 8.11

Autocorrelation values for typical block motion fields (*Foreman* at 25 fps). (Adapted from Ref. [5].)

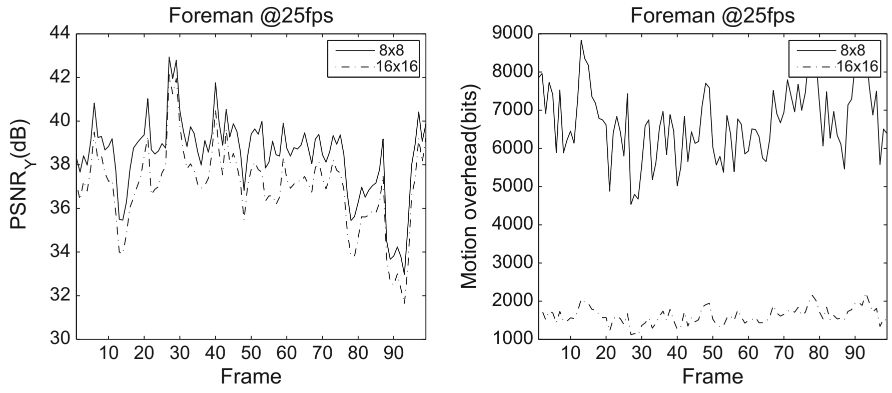


FIGURE 8.12

The effect of block size on motion estimation performance.

Example 8.2 (Full search motion estimation)

Given the following 2 × 2 block in the current frame **S**, and the corresponding search window **W**, determine the motion vector, **d**, for this block. Use a full search strategy with an SAD error metric.

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

Solution. Consider, for example, the candidate motion vector $\mathbf{d} = [0, 0]$. In this case the displaced frame difference signal is given by:

$$\mathbf{E} = \begin{bmatrix} 3 & 9 \\ 1 & 4 \end{bmatrix} - \begin{bmatrix} 1 & 3 \\ 7 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 6 \\ -6 & 3 \end{bmatrix}$$

and the SAD = 17. Similarly for all other $[d_x, d_y]$ offsets:

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

Hence the motion vector for this block is $\mathbf{d} = [1, 0]$.

The effect of search range

The *search grid* size is dictated by the maximum motion vector offset from the current block location, denoted as d_m. The value of d_m also has an influence on complexity and accuracy, since full search complexity will grow with its square (there are $(2d_m + 1)^2$ candidates to be evaluated) and most fast search methods grow with it linearly. A small value of d_m will mean that fast moving objects will not be captured in the search widow, resulting in poor prediction quality and motion failure. In contrast, a large value of d_m may be inefficient and lead to excessive computational complexity.

The choice of search window size is dictated by content type and video format. For example, sports content will normally require a larger window than talk shows, and HDTV will require a larger window than SDTV for the same frame rate. Figure 8.13 shows the effect of search window size for the CIF Foreman sequence at 25 fps. It can be seen that a small ± 5 window performs satisfactorily most of the time in this case, except for a few temporal regions (around frames 12, 50, 80, 90) where higher motion causes a significant drop in quality. The larger search window will cope with these fast motions, but does require a small increase in bit rate in order to deal with the increased average motion vector size.

The motion residual error surface

The error surface for each block (i.e. the BDM figures for each candidate vector in the search grid) will normally be multimodal, containing one or more local minima. This can be caused by periodicities in local textures, the presence of multiple motions within the block, or just a poor fit between the motion model and the apparent motion. In addition, the global minimum and maximum SAD values can vary enormously, even across a single frame in a sequence. An example of an error surface is shown in Figure 8.14 together with its corresponding current block (top right) and search

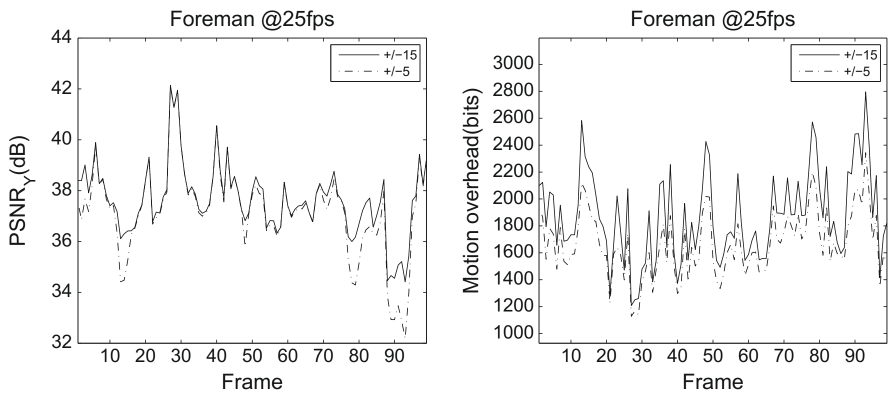


FIGURE 8.13

The influence of search range on prediction accuracy and coding bit rate.

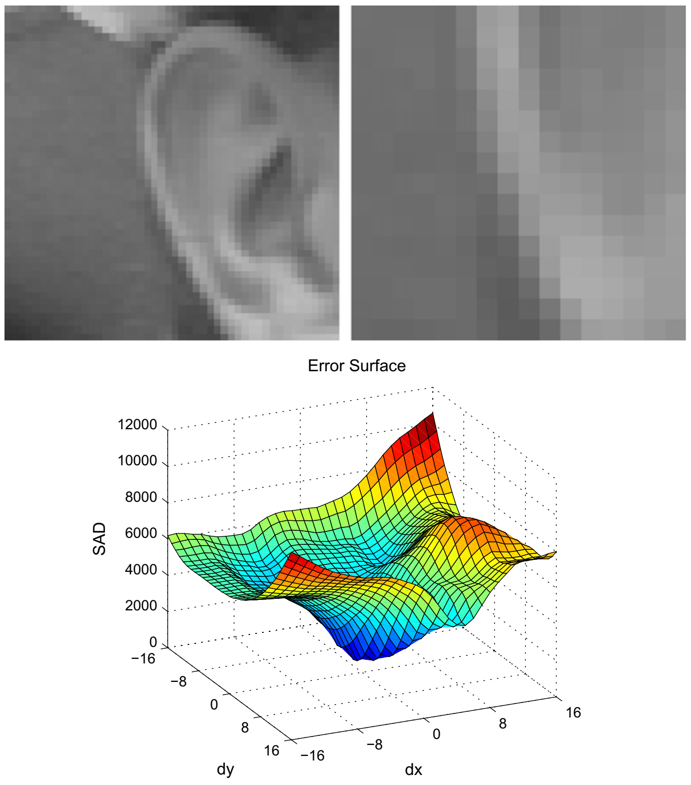


FIGURE 8.14

Typical motion residual error surface for a 16×16 block from the *Foreman* sequence (CIF at 25 fps, $\pm 16 \times \pm 16$ search window). Top left: search window. Top right: current block.

window (top left). We can observe that in the example shown, the SAD surface is indeed multimodal and extends in value from below 1000 to over 11,000.

Motion vector probabilities

The distribution of the block motion field is center-biased, implying that smaller displacements are more probable than larger ones and that the [0,0] vector is (in the absence of global motion) the most likely. This property is illustrated in Figure 8.15, for a range of sequences coded at 25 fps. The peak in the pdf at [0,0] is clear for the *Akiyo* and *Football* sequences, whereas there is an obvious offset due to the camera pan in *Coastguard*. The low activity sequence, *Akiyo*, shows a large peak in the pdf at [0,0] with very small values elsewhere. The *Football* sequence on the other hand shows a much lower peak together with a clear element due to camera panning and a significant pdf energy away from [0,0] due to the complex nature of the scene. The *Coastguard* sequence shows a wider main peak, biased due to the camera pan,

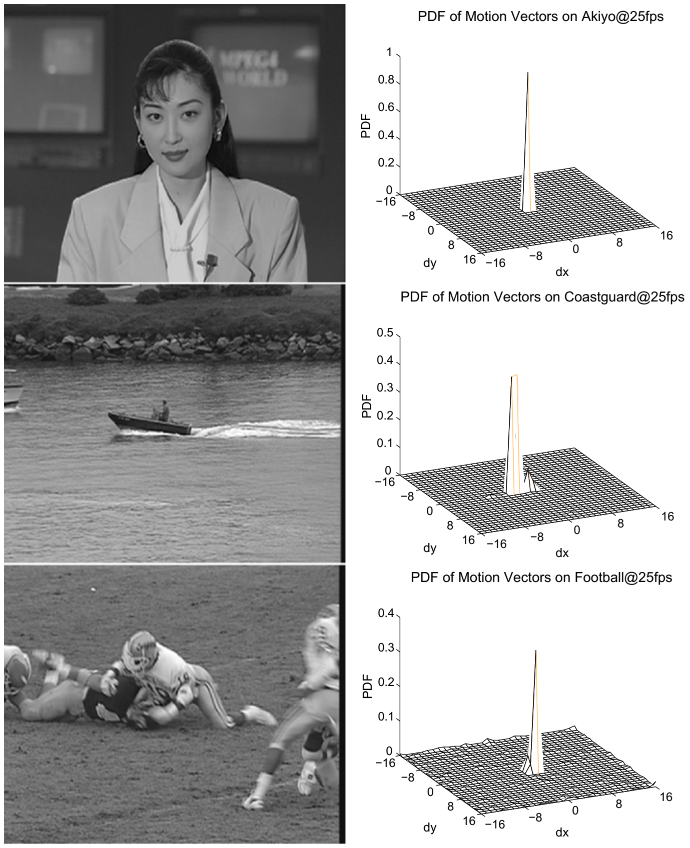


FIGURE 8.15

Motion vector pdfs for *Akiyo*, *Coastguard* and *Football* sequences.

but also energy away from [0,0], largely due to the difficulty of estimating the random movements of the dynamic water texture.

8.3.5 Motion failure

Although the block-based translational motion model works well in most cases and can provide excellent rate–distortion performance with low complexity, there are situations where the motion characteristics are such that it fails badly. Such a situation is shown in Figure 8.16. These two frames, from *Football*, capture the high activity typical in such content and the DFD, even after sub-pixel motion estimation, shows significant artifacts. These artifacts will cause the DFD to have a very low autocorrelation coefficient and hence the decorrelating transform (DCT or otherwise) will perform poorly.

So how do we deal with this situation? Basically two approaches can be used, independently or in combination. The first compares the rate–distortion performance

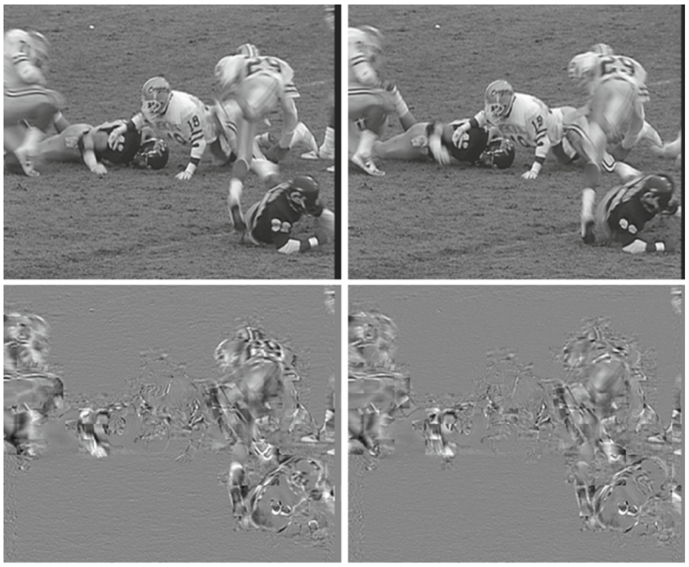


FIGURE 8.16

Example of motion model failure for *Football* sequence. Top left: current frame, right: reference frame. Bottom left: full-pixel estimation, right: half-pixel estimation.

of intra- and inter-frame coding and chooses the optimum coding mode. The second uses an in-loop deblocking filter to remove introduced edge artifacts in the reference frame. These methods are discussed further in Chapters 9 and 10.

8.3.6 Restricted and unrestricted vectors

In the case of blocks situated at or near the edge of a frame, a portion of the search area will fall beyond the frame border. In standard motion estimation, vectors are restricted to fall within the frame. However, benefits can be obtained, especially in the case of camera motion, by artificially extending the search window through border pixel extrapolation, thus supporting the use of *unrestricted motion vectors*. External pixels are normally based on a zero order or symmetric extrapolation of the border values.

8.4 Reduced complexity motion estimation

8.4.1 Pixel grids and search grids

Before we look at advanced search techniques, it is worth spending a little time differentiating between *pixel grids* and *search grids*. Both represent the region of support for motion search but we will use the latter as the basis for describing the operation of search methods from now on. The pixel grid is simply the image matrix where each element represents the pixel value within the search window. In contrast, the search grid represents the search window in terms of its offsets. Without loss of generality,

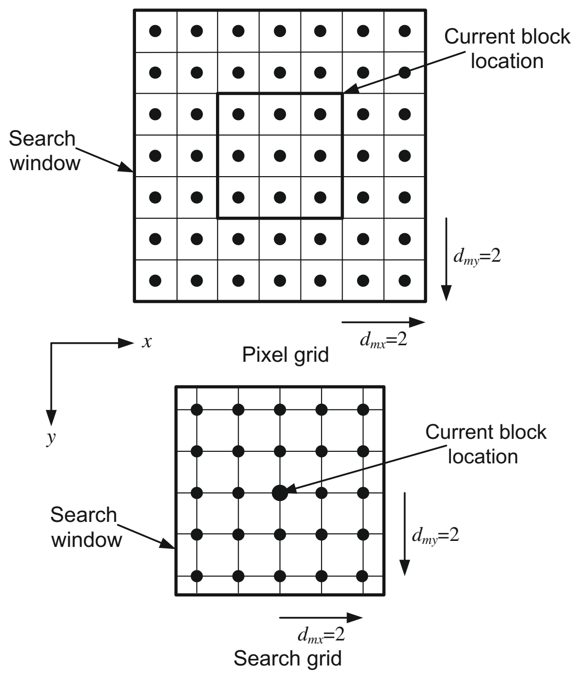


FIGURE 8.17

The pixel grid and the search grid.

assuming a square search window, the dimensions of the search grid are $2d_m + 1$ by $2d_m + 1$. This corresponds, for an N by N block size, to a search window with dimensions $2d_m + N$ by $2d_m + N$. These two representations are illustrated in Figure 8.17.

8.4.2 Complexity of full search

For each motion vector, there are $(2d_m + 1)^2$ motion vector candidates within the search grid. At each search location MN pixels are compared, where each comparison requires three operations (Sub, Abs, Add). The total complexity (in terms of basic arithmetic operations) per macroblock is thus $3MN(2d_m + 1)^2$.

Thus, for a picture resolution of $I \times J$ and a frame rate of F fps, the overall complexity is (in operations per second) given by equation (8.22):

$$C_{FS} = 3IJF(2d_m + 1)^2 \tag{8.22}$$

Example 8.3 (Full search complexity)

Consider a video sequence with the following specifications: $M = N = 16$, $I = 720$, $J = 480$, $F = 30$, $d_m = 15$. Compute the complexity of a full search motion estimation algorithm for this sequence.

Solution. As indicated above the full search complexity is given by $3IJF(2d_m+1)^2$ operations per second. Substituting the values for the given sequence gives a processor loading of 29.89×10^9 operations per second!

8.4.3 Reducing search complexity

As we will see shortly, search complexity can be reduced considerably but usually at the expense of sub-optimum performance. Many approaches exist and these can be based on:

1. **A reduced complexity BDM:** In practice it is difficult to improve upon the use of SAD which already offers a very good trade-off between complexity and performance. One interesting approach in this category is that of Chan and Siu [9] where the number of pixels used in the BDM evaluation is varied according to the block activity.
2. **A sub-sampled block motion field:** This approach exploits the smoothness properties of the block motion field. Only a fraction of the block candidates are evaluated and the remainder are interpolated before selecting the minimum BDM. For example, Liu and Zaccarin [10] use a checkerboard pattern to evaluate the motion vectors for half of the blocks. This reduces search complexity by a factor of 2 with little decrease in prediction quality.
3. **Hierarchical search techniques:** This class of method uses a multiresolution approach to progressively refine motion vector estimates across finer scales. An example of such methods, using a mean pyramid, is presented by Nam et al. [11]. Such approaches are also reported to provide more homogeneous block motion fields with a better representation of the true motion.
4. **A reduced set of motion vector candidates:** This method restricts the number of blocks evaluated using some form of directed search pattern. Most fast ME algorithms in this category are based on the assumption of a smooth and unimodal error surface (i.e. the error surface increases monotonically with distance from the minimum BDM point). We have already seen that this assumption is not always valid and algorithms can get trapped in local minima (see Figure 8.14) so the search pattern used must take this into account. The earliest approach of this type was the Two-Dimensional Logarithmic (TDL) search proposed by Jain and Jain [12], and many variants have followed. This class of method is widely considered to offer an excellent compromise between complexity and prediction quality and is commonly used in practice. Selected approaches from this category are described in the following subsections.
5. **Intelligent initialization and termination:** Again, these exploit the properties of the block motion field. Intelligent initialization criteria are used to start the search at the location of a predicted motion vector rather than at the center of the search grid. Such methods can be used with (4) to help avoid local minima.

The reader is directed to Ref. [5] for a detailed comparison of these methods. In the following sections we examine some of the more popular techniques in more detail.

8.4.4 2-D logarithmic search (TDL)

The TDL search was introduced by Jain and Jain in 1981 [12] as a method of reducing the number of motion vector candidates evaluated. The search patterns used in this method are shown in Figure 8.18 and are in the shape of a + with five points, except for the final stage where a nine-point square pattern is checked. The largest pattern is initially located at the center of the search grid, all five points are evaluated and the one with the lowest BDM is selected as the center of the next search. The process is repeated until the minimum point remains at the center, in which case the search pattern radius is halved. This process continues until the radius is equal to 1, when the nine-point square pattern is invoked. This process is described in Algorithm 8.1.

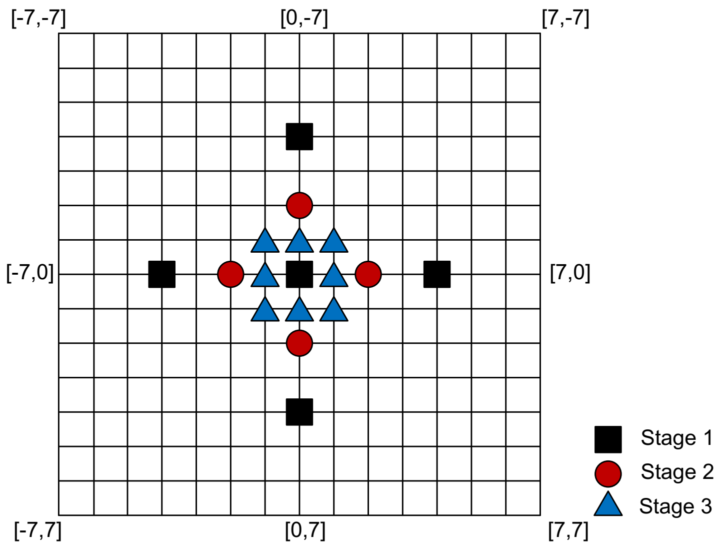


FIGURE 8.18

2-D logarithmic search patterns.

Algorithm 8.1 TDL search.

```
1. Set initial search radius $r = 2^{k-1}$where $k = \lceil \log_2(d_m) \rceil$;
2. Set checking points to: $\mathbf{\Gamma} = \{[0,0], [0, \pm r][\pm r, 0]\}$;
3. Evaluate the BDM at the five candidate locations on the cross and select the grid
   point with the lowest BDM: $\mathbf{d}' = \arg\left(\min_{(i,j)\in\Gamma}(\text{BDM}(i,j))\right)$;
4. IF the minimum remains at the center of the pattern THEN $r = \frac{r}{2}$;
5. IF $r > 1$ THEN $\mathbf{\Gamma} = \mathbf{d}' + \{[0,0], [0, \pm r][\pm r, 0]\}$, GO TO (3);
6. IF $r = 1$ THEN invoke nine-point square pattern with $\mathbf{\Gamma} = \mathbf{d}' + \{[0,0], [\pm r, \pm r],$
   $[0, \pm r][\pm r, 0]\}$, $\mathbf{d} = \arg\left(\min_{(i,j)\in\Gamma}(\text{BDM}(i,j))\right)$, STOP.
```

8.4.5 *N*-Step Search (NSS)

The N-Step Search (NSS), introduced by Koga et al. [13], is similar to the TDL search. It uses a uniform pattern comprising nine evaluation points as shown in Figure 8.19, where the search radius decreases by a factor of 2 at each iteration. As for the TDL search, the process terminates when the search radius is equal to 1. The advantage of this approach is that it requires a fixed number of iterations, so offers regularity in terms of hardware or software implementation. The method is described in Algorithm 8.2. A worked example of a Three-Step Search (TSS) is given in Example 8.4.

The complexity of the NSS algorithm is easy to compute and is:

$$C_{NSS} = 3IJF(8k + 1) \tag{8.23}$$

where k is the number of steps as indicated in Algorithm 8.2.

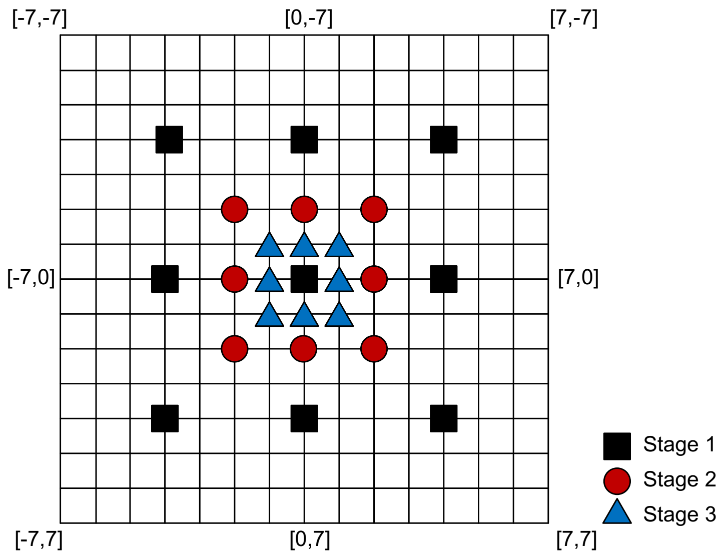


FIGURE 8.19

N-Step Search patterns.

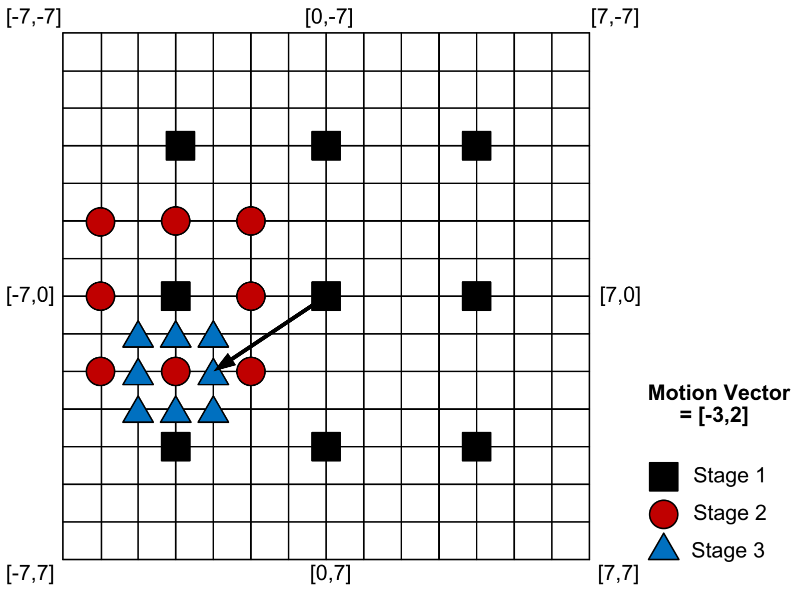
Algorithm 8.2 NSS search.

1. Set initial search radius $r = 2^{k-1}$where $k = \lceil \log_2(d_m) \rceil$;
2. Set checking points to: $\mathbf{\Gamma} = \{[0, 0], [\pm r, \pm r], [0, \pm r], [\pm r, 0]\}$;
3. Evaluate the BDM at the nine candidate locations and select the grid point with the lowest BDM: $\mathbf{d}' = \arg\left(\min_{(i,j)\in\Gamma} (\text{BDM}(i, j))\right)$;
4. Set $r = \frac{r}{2}$;
5. IF $r < 1$ THEN $\mathbf{d} = \mathbf{d}'$, STOP; ELSE set new search locations: $\mathbf{\Gamma} = \mathbf{d}' + \{[0, 0], [\pm r, \pm r], [0, \pm r], [\pm r, 0]\}$; GO TO (3).

Example 8.4 (The three-step search)

Assuming a 15 × 15 search grid and a monotonically decreasing error surface with a single minimum, demonstrate how the TSS search algorithm would converge to a motion vector of $\mathbf{d} = [-3, 2]$.

Solution. The NSS algorithm will always converge in N steps, but the convergence profile will depend on the actual shape of the error surface. One possible solution is shown in the figure below.



8.4.6 Diamond search

The Diamond Search (DS) was first introduced by Zhu and Ma in 1997 [14] and has demonstrated faster processing, with similar distortion, in comparison with TSS and NTSS approaches. DS adopts two search patterns—one large diamond for general gradient search and a second smaller diamond for final stage refinement. The approach is again similar to the TDL search in that the large diamond pattern at stage k is centered on the point with minimum BDM from stage $k - 1$. When the search minimum remains at the center of the pattern, the small diamond pattern is invoked to refine the motion vector before termination. The two diamond patterns are shown in Figure 8.20 and the process is described in Algorithm 8.3. An illustration of the diamond search process is given in Example 8.5. It should be noted that the complexity of DS is dependent on the orientation of the optimum motion vector and so is content dependent.

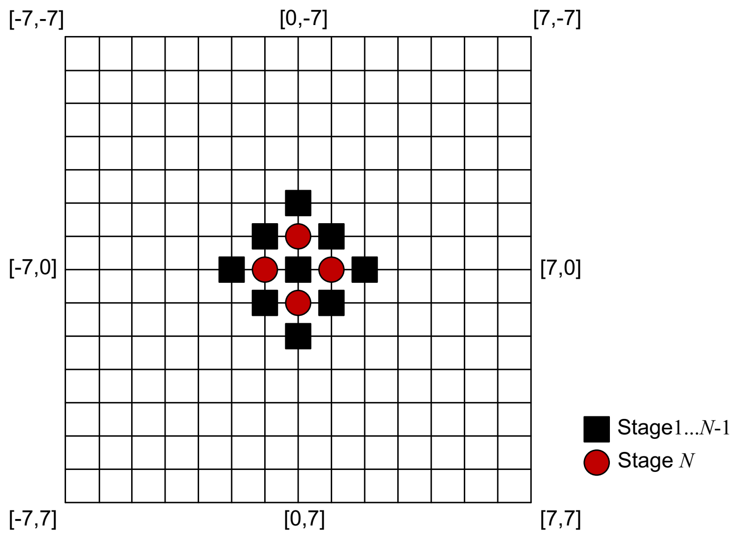


FIGURE 8.20

Diamond search patterns.

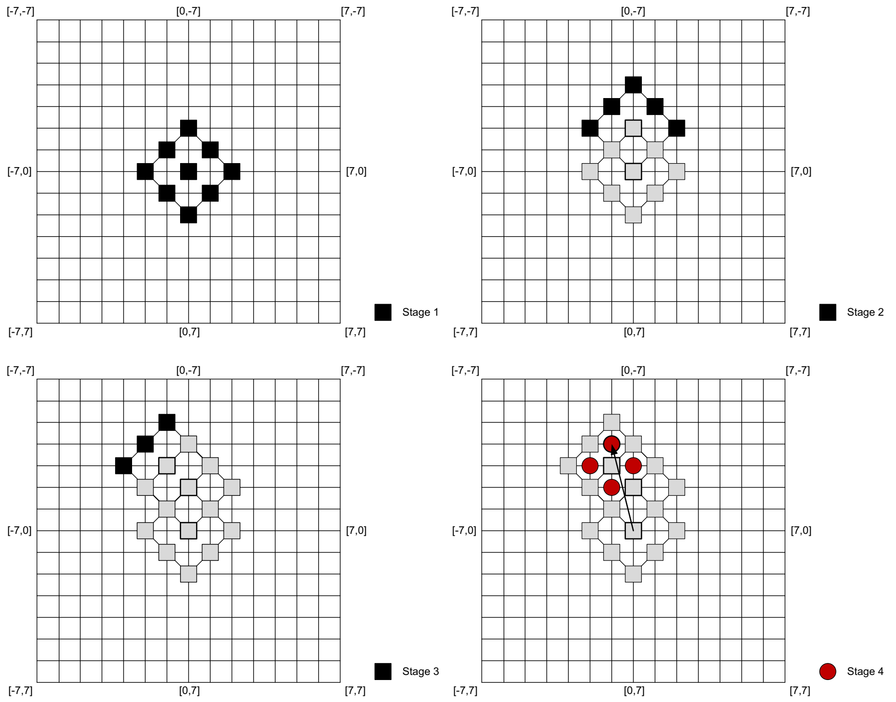
Algorithm 8.3 Diamond search.

```
1. Initialize motion vector estimate: $\mathbf{d}' = [0, 0]$;
2. Set checking points to: $\boldsymbol{\Gamma} = \mathbf{d}' + \{[0, 0], [0, \pm 2][\pm 2, 0][\pm 1, \pm 1]\}$;
3. Evaluate the BDM at the nine candidate locations on the diamond and select the
   grid point with the lowest BDM: $\mathbf{d}' = \arg\left(\min_{(i,j)\in\Gamma}(\mathrm{BDM}(i, j))\right)$;
4. IF the minimum remains at the center of the pattern THEN $\boldsymbol{\Gamma} = \mathbf{d}' + \{[0, 0],$
   $[0, \pm 1], [\pm 1, 0]\}$, $\mathbf{d} = \arg\left(\min_{(i,j)\in\Gamma}(\mathrm{BDM}(i, j))\right)$, STOP;
5. GO TO (2).
```

Example 8.5 (The diamond search)

Assuming a 15×15 search grid and a monotonically decreasing error surface with a single minimum, demonstrate how the DS search algorithm would converge to a motion vector of $\mathbf{d} = [-1, -4]$.

Solution. The DS algorithm will not converge in a specific number of steps but will track the gradient of the error surface using the large diamond shape until the minimum stays at the center of the diamond (stages 1–3 in this example). The small diamond shape is then introduced at stage 4 to identify the final motion vector. The convergence profile will of course depend on the actual shape of the error surface. One possible solution is shown in the following figures:



8.4.7 Hexagonal search

There are many variants on the methods presented above. One worthy of note is the hexagonal search (HEXBS), which again uses two pattern types, but this time in the shape of a hexagon (Figure 8.21). HEXBS was introduced by Zhu et al. [15] and, like DS, the smaller pattern is used only once for refinement at the final iteration. HEXBS has the advantage over diamond search that it offers greater orientation flexibility with the number of new checking points introduced at each iteration of the gradient descent stage being constant (three new points).

The advancing speed for DS is 2 pixels per step horizontally and vertically, and $\sqrt{2}$ pixels per step diagonally, whereas the HEXBS has horizontal and vertical advancements of 2 pixels per step horizontally and vertically and $\sqrt{5}$ pixels per step in the direction of the off-axis search points. This means that HEXBS is more consistent in its search performance and can outperform DS by up to 80% for some motion vector locations.

The complexity of the HEXBS algorithm can easily be seen to be:

$$C_{\text{HEXBS}} = 7 + 3k + 4 \tag{8.24}$$

where k is the number of iterations required to reach the minimum of the error surface.

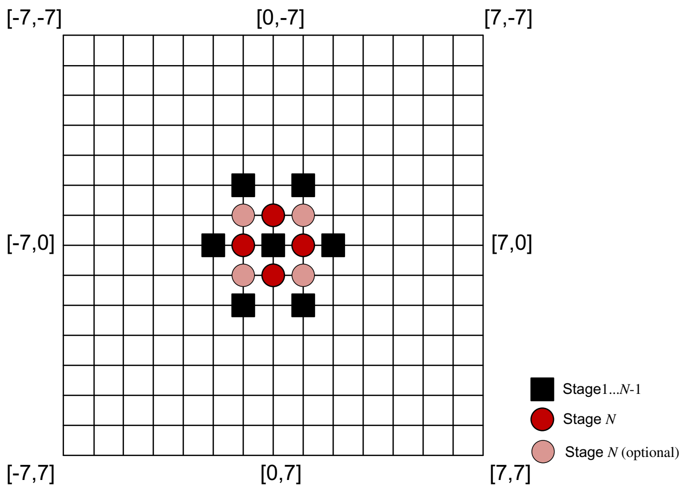


FIGURE 8.21

Hexagonal search patterns.

8.4.8 Initialization and termination criteria

Initialization

The use of intelligent initialization and termination criteria can, in many cases, improve search performance and help to avoid the problems of the search becoming trapped in local minima. Referring to Figure 8.22, which shows a current block P and its near causal neighbors A–D, a range of candidate initialization vectors can be formulated, based on the local smoothness property of the block motion field. These include:

$$\hat{\mathbf{d}}_P = \{[0,0], \mathbf{d}_A, \mathbf{d}_B, \mathbf{d}_C, \mathbf{d}_D, \text{med}(\mathbf{d}_A, \mathbf{d}_C, \mathbf{d}_D), \text{med}(\mathbf{d}_A, \mathbf{d}_B, \mathbf{d}_C)\} \tag{8.25}$$

An initialization algorithm would typically evaluate some or all of the candidate vectors from equation (8.25), choose that with the lowest BDM, and start the search at $\hat{\mathbf{d}}_P$ rather than always at [0,0].

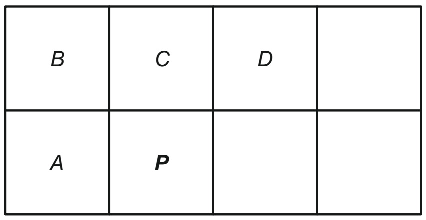


FIGURE 8.22

Predicted motion vectors for initialization.

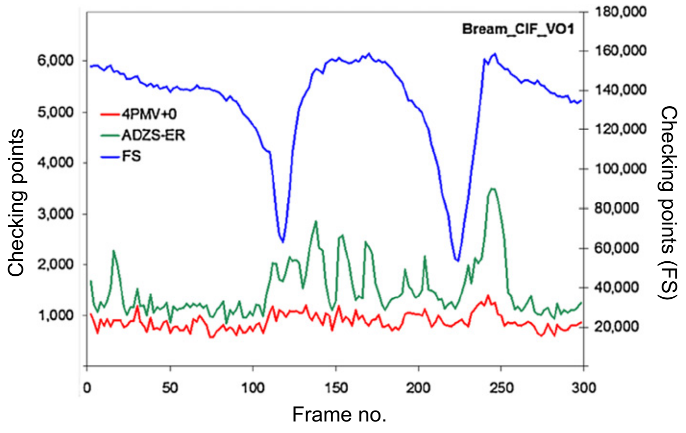


FIGURE 8.23

Fast motion estimation performance comparison for the CIF Bream sequence at 25 fps: 4PMV+0 vs ADZS-ER vs FS.

Termination

The use of termination criteria is largely dictated by the implementation method used. For example, in the case of a software implementation, a search could be terminated by setting an acceptable block distortion threshold BDM_{Th} and exiting the search when $BDM \leq BDM_{Th}$. Furthermore, inside each SAD evaluation loop, execution could be terminated when the distortion measure for the current offset $[i, j]$ exceeds the current minimum. i.e. when $BDM_{i,j} \geq BDM_{opt}$, or when the rate–distortion cost is sufficiently small.

Several methods have been reported that provide significant speed-up using intelligent initialization and termination. These include the work of Tourapis et al. [16,17] on PVMFast and EPZS and the work in the author's lab by Sohm [18]. For example, Figure 8.23 shows complexity results including the 4PMV+0 algorithm, Diamond Search and Full Search.

8.4.9 Reduced complexity block matching algorithm performance comparisons

Several authors have published tables comparing motion estimation methods in terms of complexity and prediction accuracy (see, for example, Ref. [16]). Table 8.2 compares some of the algorithms presented in this chapter. It can be seen that methods such as the Simplex Search [19] provide the most accurate results (closest to FS) with a competitive complexity. Techniques adopting intelligent initialization and termination criteria such as 4PMV+0 on the other hand require extremely low numbers of checking points with only a modest reduction in prediction accuracy.

Table 8.2 Reduced complexity BMME comparisons for the Bream CIF sequence at 25 fps. Top: checking points per macroblock. Bottom: SAD error relative to full search.

Algorithm	4PMV + 0	ADZS	DS	Simplex	NSS	FS
Minimum	1	1	13	9	32.9	1002.4
Average	4.1	5.9	16.1	14.2	33.0	1018.3
Maximum	9.6	15.3	22.2	21.8	33.1	1024.0
Average	4.248	4.492	4.311	4.078	4.494	3.975
% vs FS	106.9	113.0	108.5	102.6	113.1	100.0

8.5 Motion vector coding

8.5.1 Motion vector prediction

Standards such as H.264/AVC and HEVC have made substantial progress in improving rate–distortion performance. However, as a consequence, the motion coding overhead has proportionally increased, with over 50% of the bits being dedicated to motion for some sequences with high QP values.

Coding of motion vectors is normally performed predictively to exploit the smoothness property of the block motion field. Consider coding the current block P in Figure 8.22. A predicted motion vector can be generated using equation (8.26). If block D is not available then block B can be used instead.

$$\hat{\mathbf{d}}_P = \{\mathrm{med}(\mathbf{d}_A, \mathbf{d}_C, \mathbf{d}_D)\} \tag{8.26}$$

The motion vector difference or residual is then computed and entropy coded for transmission:

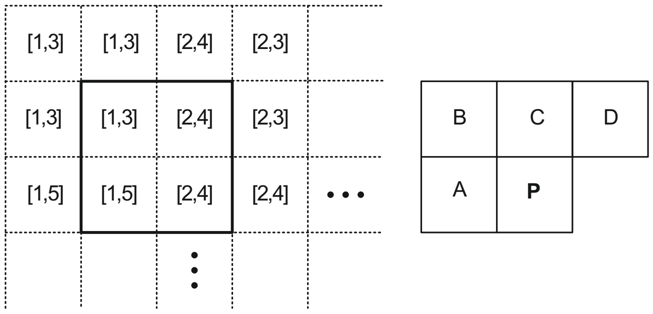
$$\mathbf{e}_P = \mathbf{d} - \hat{\mathbf{d}}_P \tag{8.27}$$

8.5.2 Entropy coding of motion vectors

The motion vector prediction process will result in a decorrelated residual signal that will mainly comprise small values. In the same way as we used entropy coding based on Huffman or arithmetic methods in Chapter 7 to code quantized transform coefficients, so too can we apply this to motion vector residuals. Clearly, different VLC tables will be required for Huffman coding and different probability distributions for arithmetic coding.

Example 8.6 (Motion vector coding)

Motion vectors for four adjacent blocks in two partial rows of a frame are shown below, outlined in bold. Using the prediction scheme: $\hat{\mathbf{d}}_P = \mathrm{med}(\mathbf{d}_A, \mathbf{d}_C, \mathbf{d}_D)$, compute the predicted motion vectors for each of these blocks together with their residuals.



Solution. The predicted motion vectors and residuals for each of the four blocks outlined (labeled top left to bottom right) are given in the table below:

Block, P	**1**	**2**	**3**	**4**
$\hat{\mathbf{d}}_P$	[1,3]	[2,3]	[1,4]	[2,4]
$\mathbf{e}_P$	[0,0]	[0,1]	[0,1]	[0,0]

Note that the prediction process described here must be modified for boundary blocks. This is described further in Chapter 12.

8.6 Summary

This chapter has addressed the important topic of motion estimation, the primary technique for reducing temporal correlation in an image sequence. We saw how temporal correlations can be exploited to provide additional coding gain. Most of the chapter focused on block matching and block-based motion estimation using a translational motion model. This approach is in widespread use and has been adopted in all standardized video codecs to date. The chapter also described a number of methods for reducing the complexity of motion estimation, comparing a range of fast search methods alongside descriptions of intelligent initialization and termination techniques.

In the next chapter we will show how to integrate the motion estimation algorithm into a DPCM prediction loop at the encoder. We also examine a range of methods for improving still further the coding gain of this block-based hybrid codec.

References

[1] T. Ebrahimi, M. Kunt, Visual data compression for multimedia applications, Proceedings of the IEEE 86 (6) (1998) 1109–1125.

[2] C. Glasbey, K. Mardia, A review of image warping methods, Journal of Applied Statistics 25 (1998) 155–171.

[3] F. Zhang, D. Bull, A parametric framework for video compression using region-based texture models, IEEE Journal of Selected Topics in Signal Processing 6 (7) (2011) 1378–1392.

[4] R. Vigars, A. Calway, D. Bull, Context-based video coding, in: Proceedings of the IEEE International Conference on Image Processing, 2013, pp. 1953–1957.

[5] M. Al-Mualla, C. Canagarajah, D. Bull, Video Coding for Mobile Communications, Academic Press, 2002.

[6] A. Netravali, J. Robins, Motion compensated television coding: part 1, Bell Systems Technical Journal 58 (1979) 631–670.

[7] D. Kuglin, D. Hines, The phase correlation image alignment method, in: Proceedings of the IEEE International Conference on Cybernetics and Society, 1975, pp. 163–165.

[8] G. Thomas, Television Motion Measurement for DATV and Other Applications, BBC Research Technical Report 1987/11, 1987.

[9] Y. Chan, W. Siu, New adaptive pixel decimation for block vector motion estimation, IEEE Transactions on Circuits and Systems for Video Technology 6 (1) (1996) 113–118.

[10] B. Liu, A. Zaccarin, New fast algorithms for the estimation of motion vectors, IEEE Transactions on Circuits and Systems for Video Technology 3 (2) (1993) 148–157.

[11] K. Nam, J. Kim, R. Park, Y. Shim, A fast hierarchical motion vector estimation algorithm using mean pyramid, IEEE Transactions on Circuits and Systems for Video Technology 5 (4) (1995) 344–351.

[12] J. Jain, A. Jain, Displacement measurement and its application in interframe image coding, IEEE Transactions on Communications 29 (12) (1981) 1799–1808.

[13] T. Koga, K. Linuma, A. Hirano, T. Ishiguro, Motion compensated interframe coding for video conferencing, in: Proceedings of the National Telecommunications Conference, 1981, pp. G5.3.1–G5.3.5.

[14] S. Zhu, K. Ma, A new diamond search algorithm for fast block matching motion estimation, IEEE Transactions on Image Processing 9 (2) (2000) 287–290.

[15] C. Zhu, X. Lin, L. Chau, Hexagon-based search pattern for fast block motion estimation, IEEE Transactions on Circuits and Systems for Video Technology 12 (5) (2002) 255–349.

[16] A. Tourapis, O. Au, M. Liou, Predictive motion vector field adaptive search technique (PMVFAST): enhancing block-based motion estimation, in: Proceedings of the SPIE Visual Communications and Image Processing, vol. 4310, 2001, pp. 883–893.

[17] A. Tourapis, Enhanced predictive zonal search for single and multiple frame motion estimation, in: Proceedings of the SPIE Visual Communications and Image Processing, vol. 4671, 2002, pp. 1069–1078.

[18] O. Sohm, Method for Motion Vector Estimation, US Patent 7,260,148, 2007.

[19] 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.