vertical component to the enhancement filter, making the overall filter separable with 3×3 support. ■

11.2 MOTION ESTIMATION AND MOTION COMPENSATION

Motion compensation (MC) is very useful in video filtering to remove noise and enhance signal. It is useful since it allows the filter or coder to process through the video on a path of near-maximum correlation based on following motion trajectories across the frames making up the image sequence or video. Motion compensation is also employed in all distribution-quality video coding formats, since it is able to achieve the smallest prediction error, which is then easier to code. Motion can be characterized in terms of either a velocity vector $\boldsymbol{v}$ or displacement vector $\boldsymbol{d}$ and is used to warp a *reference frame* onto a *target frame*. Motion estimation is used to obtain these displacements, one for each pixel in the target frame.

Several methods of motion estimation are commonly used:

- Block matching
- Hierarchical block matching
- Pel-recursive motion estimation
- Direct optical flow methods
- Mesh-matching methods

Optical flow is the apparent displacement vector field $\boldsymbol{d} = (d_1, d_2)$ we get from setting (i.e., forcing) equality in the so-called *constraint equation*

$$x(n_1, n_2, n) = x(n_1 - d_1, n_2 - d_2, n - 1). \qquad (11.2\text{–}1)$$

All five approaches start from this basic equation, which is really just an idealization. Departures from the ideal are caused by the covering and uncovering of objects in the viewed scene, lighting variation both in time and across the objects in the scene, movement toward or away from the camera, as well as rotation about an axis (i.e., 3-D motion). Often the constraint equation is only solved approximately in the least-squares sense. Also, the displacement is not expected to be an integer as assumed in (11.2–1), often necessitating some type of interpolation to be used.

Motion cannot be determined on a pixel-by-pixel basis since there are two components for motion per pixel, and hence twice the number of unknowns as equations. A common approach then is to assume the motion is constant over a small region called the *aperture*. If the aperture is too large, then we will miss detailed motion and only get an average measure of the movement of objects in our scene. If the aperture is too small, the motion estimate may be poor to very wrong. In fact, the so-called *aperture problem* concerns the motion estimate in the square region shown in Figure 11.2–1.

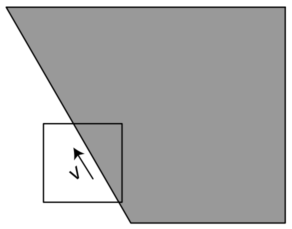

FIGURE 11.2–1

Illustration of the *aperture problem* with the square indicating the aperture size.

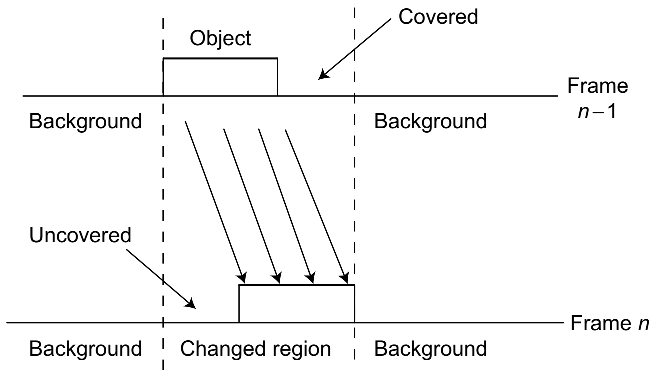

FIGURE 11.2–2

Illustration of covering and uncovering of background by an object moving in the foreground.

If the motion of the uniform dark region is parallel to its edge, then this motion cannot be detected. Since this situation would typically only hold for small regions in natural images, the aperture effect leads us to choose a not-too-small aperture size. Thus, finding the right aperture size is an important problem that depends on the video content.

Another issue is *covering* and *uncovering*, as illustrated in Figure 11.2–2, showing a 1-D depiction of two successive frames n and $n-1$, with an object moving to the right. We assume a simple object translating in the foreground over a fixed background, not an unreasonable local approximation of video frames. We see that part of the background region in target frame n is uncovered, while part of the background region in reference frame $n-1$ is covered. Motion estimation that tries to match regions in the two frames will not be able to find good matches in either the covered or uncovered regions. However, within the other background regions, matches should be good and matching should also be good within a textured object, at least if

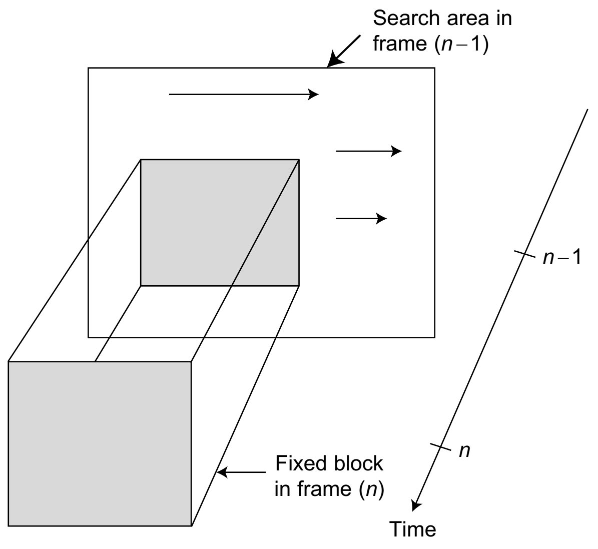

FIGURE 11.2–3

Illustration of simple block matching.

it moves in a trackable way, and the pixel samples are dense enough. The problem in the relatively small covered/uncovered regions is that there are two motions present there.

Block-Matching Method

We intend to estimate a displacement vector at the location (n_1, n_2, n) in the target frame. In *block matching* (BM) [5], we use template matching of the block centered on this point to blocks in a specified *search area* in the reference frame, as illustrated in Figure 11.2–3, where we take the immediately prior frame as reference.

Often the search area size is given as $(\pm M_1, \pm M_2)$ and centered on the current pixel location (n_1, n_2) in the reference frame, then a total of $(2M_1 + 1)(2M_2 + 1)$ searches must be done in a *full search*, and this must be done for each pixel where the motion is desired. Often the block matching is not conducted at every pixel in the target frame and an interpolation method is used to estimate the motion in between these points. Common error criteria are *mean-square error* (MSE), *mean-absolute error* (MAE),[2] or even number of pixels in the block actually disagreeing for discrete-amplitude or digital data.

[2]Actually preferred in practical applications because MAE works a bit better than MSE owing to being more robust, and it only involves additions. MAE goes by other acronyms, too: mean absolute difference (MAD) and sum absolute difference (SAD).

We express the MSE in block matching as

$$\mathcal{E}(\boldsymbol{d}) \triangleq \sum_{\boldsymbol{k}} \big(x(\boldsymbol{n}+\boldsymbol{k},n) - x(\boldsymbol{n}+\boldsymbol{k}-\boldsymbol{d},n-1)\big)^2, \qquad (11.2\text{–}2)$$

for a square block centered at position $\boldsymbol{n} = (n_1, n_2)^T$ as a function of the vector displacement $\boldsymbol{d} = (\boldsymbol{d_1}, \boldsymbol{d_2})^T$. We seek the displacement vector that minimizes this error

$$\boldsymbol{d}_o \triangleq \arg\min_{\boldsymbol{d}} \mathcal{E}(\boldsymbol{d}).$$

Since the MSE in (11.2–2) is susceptible to outliers, often the MAE is used in applications. In addition to its being less sensitive to statistical outliers, another advantage of MAE is that it is simpler to compute.

In addition to the computationally demanding full-search method, there are simpler approximations involving a much reduced number of evaluations of the error (11.2–2). The methods either involve sampling the possible locations in the search region or sampling the elements in the block when calculating (11.2–2). Two examples of the former strategy are 2-D log search and three-step search. With reference to Figure 11.2–4, the three-step search proceeds as follows: First, the search window is broken up into four quadrants, and motion vectors are tested on a 3×3 grid with corners centered in the four quadrants. Here, we illustrate a case where the lower right corner is the best match at the first step, the top right corner is best at the second step, and the top right corner is best at the third and final step. A performance comparison is shown in Figure 11.2–5 from [6].

We note that all three techniques perform much better than simple frame differencing, which equivalently treats all displacement vectors as zero. While both fast

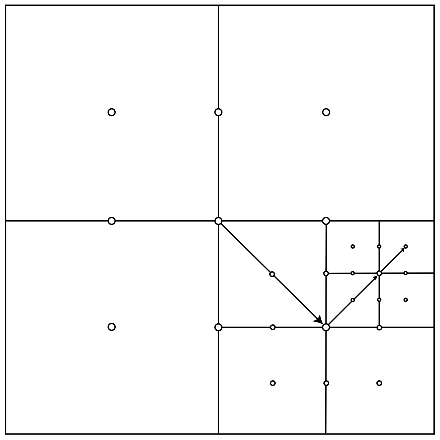

FIGURE 11.2–4

An illustration of three-step block matching.

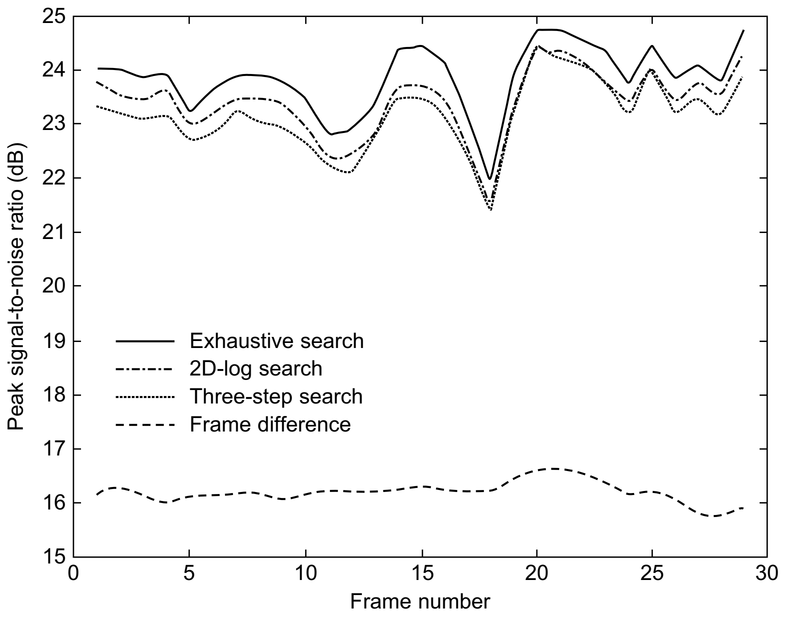

FIGURE 11.2–5

Illustration of PSNR performance of exhaustive, 2-D log, three-step search, and simple frame difference. ([6] ©1995 Academic Press)

methods cut computation by a factor of 10 or more versus full-search block matching, they can lose up to 1–2 dB in *prediction peak SNR* (PSNR), measured in decibels (dB) as

$$PSNR = 10\log_{10}\left(\frac{255^2}{\mathcal{E}(\boldsymbol{d})}\right),$$

for the 8-bit images being considered, since their peak value would be 255. For 10-bit images, the formula would substitute 1023 for the peak value.

The other class of methods to speed up block matching involve sampling in the calculation of the error (11.2–2). So the amount of computation in evaluating the distortion for each searched location is reduced. Liu and Zaccarin [7] presented a method involving four phases of subsampling and alternated subsampling patterns among the blocks, while using the MAE error criterion. This method achieved approximately a four-fold reduction in the amount of computation, but only a slight increase in the average prediction MSE.

There are some unavoidable problems with the block-matching approach. A small block size can track small moving objects, but the resulting displacement estimate is then sensitive to image noise.[3] For example, a small block might just span a flat region in the image, where the displacement cannot be defined. A large block size is less sensitive to noise, but cannot track the motion of small objects. Similarly, a large search area can track fast motion but is computationally intensive. A small search area may not be large enough to catch or track the real motion.

The best matching block is often good enough for a block-based predictive video compression, where bits can be spent coding the prediction residual. However, in video filtering, when the estimated motion is not the true physical motion, visible artifacts will often be created. Hence, we need an improvement on the basic block-matching method for the filtering application. The same is true for MC interpolation, frame-rate conversion, and pre- and postprocessing in video compression. Also, some highly scalable video coders use MC in a temporal filtering structure to generate lower frame-rate versions of the original video. Highly accurate motion vectors are important in this case too. A variation of block matching, called hierarchical block matching, can achieve a much better estimate of the "true motion."[4]

Hierarchical Block Matching

The basic idea of hierarchical algorithms is to first estimate a coarse motion vector at low spatial resolution. Then this estimated motion is refined by increasingly introducing higher spatial frequencies. Both subband/wavelet pyramids and Gaussian pyramids have been used for this purpose. An often cited early reference on *hierarchical block matching* (HBM) is Thoma and Bierling [8].

We start by creating a spatial pyramid, with resolution levels set at a power of two. Typically three or four stages of resolution are employed. We start at the lowest resolution level (highest pyramid level) and perform simple block matching. It has been found helpful to have the block size there agree with the average size of the largest moving areas in the video. Next, we start down the pyramid, increasing the resolution by the factor 2×2 at each level. We double the displacement vector from the previous level to get the initial search location at the present level. We finish up at the pyramid base level, which is full resolution. Both the block sizes and the search regions can be chosen distinctly for each resolution and are generally in the range from 4×4 to 32×32. The maximum search area is usually small at each resolution (i.e., ± 2), since only refinements are needed. The search area may also be small at the initial pyramid level because of the low resolution. Thus we can expect considerable savings in complexity for HBM. Some other improvements to block matching include subpixel accuracy, *variable-size block matching* (VSBM), overlapping blocks, block prediction of motion vectors, and *hierarchical VSBM* (HVSBM).

[3]By the way, it turns out that some amount of noise is always present in real images. This comes from the common practice of setting the bit depth on sensors and scanners to reach the first one or two bits of the physical noise level, caused by photons, film grains, etc.
[4]Really it is optical flow—i.e., the apparent movement that comes from our 2-D observations.

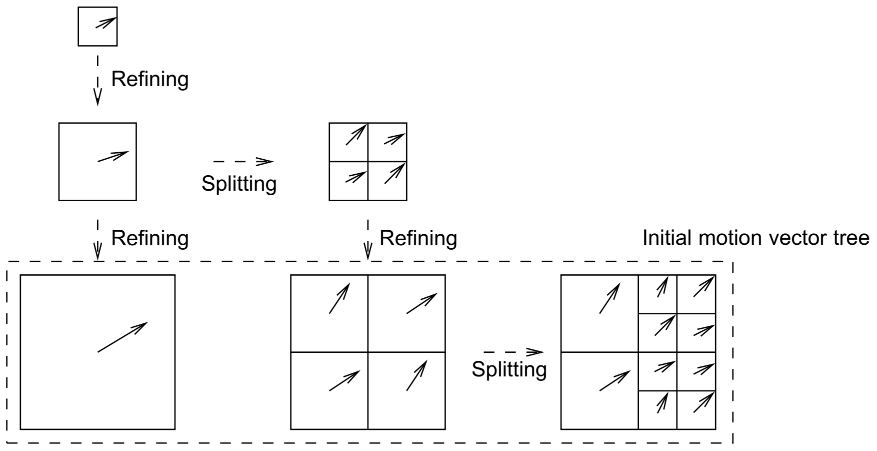

FIGURE 11.2–6

An illustration of the refining and spliting process in HVSBM.

A diagram of HVSBM is shown in Figure 11.2–6. We start with a spatial pyramid that can be obtained as a succession of LL subbands by subband/wavelet filtering and 2×2 decimation. Starting at the coarsest level, at the top of the pyramid, a block-matching motion estimation is performed. Then this displacement estimate $\boldsymbol{d}_0$ is propagated down the pyramid one level, and $2\boldsymbol{d}_0$ is used to initialize a search over a small region to *refine* the motion value to $\boldsymbol{d}_1$. At this time, if the MC error measure is too large, the block is *split* into four, and the process of refining is repeated to generate $\boldsymbol{d}_1$, and this process of refining and splitting is carried down the pyramid to the bottom, resulting in a variable size block-based motion field. In a computationally more demanding variation of HVSBM, we start at the coarsest resolution with the smallest block size, and refine this motion field to the bottom of the pyramid (i.e., the highest resolution level). Then this resulting motion field is *pruned* back by merging nodes to a variable-size block-based motion field. This can be done using the BFOS algorithm, and this *bottom-up* approach generally results in a more accurate motion field than the top-down method mentioned in the previous paragraph, but it is more computationally intensive.

In a video coding application, the error criteria can be composed of motion field error, either MSE or MAE, to weigh the increase in motion-vector rate due to the split (top-down) or decrease in motion-vector rate due to the merge (bottom-up). A Lagrangian approach is often used to control the bitrate of the motion information. More on coding motion vectors for video compression is contained in Chapter 12.

Overlapped Block Motion Compensation

The motivation to overlap the blocks used in a conventional block-matching estimate is to increase the smoothness of the resulting velocity field. This can be considered as a method to reduce the spatial frequency aliasing in sampling the underlying velocity field. While one could simply overlap the blocks used in a simple block-matching estimate, this could mean much more computation. For example, if the blocks were overlapped by 50% horizontally and vertically, it would be four times more computation if the block-matching estimation were done independently, as well as four times more velocity information to transmit in the video compression application of Chapter 12. So, effectively we are more interested in smoothing than in alias reduction, and the *overlapped block motion compensation* (OBMC) technique [9, 10] simply weights each velocity vector with four neighbor velocity estimates from the four nearest neighbor nonoverlapping blocks. Thus we effectively overlap the velocity vectors without overlapping the blocks themselves. This is usually done with a few prescribed weighting windows.

A theoretical motivation for the overlapping can be obtained from (4) of [10],

$$\begin{aligned}\widehat{x}(\boldsymbol{n}, n) &= E\{x(\boldsymbol{n}, n)\,|\,\boldsymbol{X}(n-1), \mathcal{V}_{\boldsymbol{n}}\}\\ &= \int f_{\boldsymbol{n}}(\boldsymbol{n}\,|\,\mathcal{V}_{\boldsymbol{n}})x(\boldsymbol{n}-\boldsymbol{v}\Delta t, n-1)d\boldsymbol{v},\end{aligned}$$

only slightly changed for our notation. Here, we are performing a motion-compensated estimate of frame $\boldsymbol{X}(n)$, as a conditional mean over shifted versions of frame $\boldsymbol{X}(n-1)$, with interframe interval Δt, making use of the conditional pdf $f_{\boldsymbol{n}}(\boldsymbol{v}\,|\,\mathcal{V}_{\boldsymbol{n}})$, which depends on $\mathcal{V}_{\boldsymbol{n}}$, the motion vector data sent in this and neighboring blocks. Assuming linear weights (i.e., no dependence on the data values in $\mathcal{V}_{\boldsymbol{n}}$), they obtain

$$\widehat{x}_n(\boldsymbol{n}) = \sum_{N_b(\boldsymbol{x})} w_b(\boldsymbol{n})x(\boldsymbol{n}-\boldsymbol{v}_b\Delta t, n-1), \qquad (11.2\text{–}3)$$

where the sum is over velocity vectors in the neighboring blocks $N_b(\boldsymbol{n})$. A formula for obtaining an optimized block weighting function $w_b(\boldsymbol{n})$ is given in [10]. Simple weighting windows are given there too.

The initial estimate obtained in this way can be improved upon by iteratively updating the velocity estimates from the various blocks, one at a time, by utilizing the resulting overlapped estimate (11.2–3) in the error calculation (11.2–2). OBMC is used in the H.263 video compression standard for visual conversation and has also been adapted for use in some SWT video coders. In the compression application, it can smooth the velocity field without the need to transmit additional motion vector bits, since the block overlapping can be done separately at the receiver given the transmitted motion vectors, still only one for each block. The overlapping of the blocks makes the velocity field smoother and removes the artificial blocked structure. This is especially important for SWT coders, where a blocky motion vector field could lead, through motion compensation, to a blocky prediction residual that would have false and excessively high spatial frequency information.

Pel-Recursive Motion Estimation

This iterative method recursively calculates a displacement vector for each pixel (a.k.a. pel) in the current frame. We start with an estimate $\boldsymbol{d} = (d_1, d_2)^T$ for the current displacement. Then we use the iterative method,

$$\widehat{d}_1^{(k+1)} = \widehat{d}_1^{(k)} - \epsilon \frac{\partial \mathcal{E}}{\partial d_1}|_{\boldsymbol{d}=\widehat{\boldsymbol{d}}^{(k)}},$$

$$\widehat{d}_2^{(k+1)} = \widehat{d}_2^{(k)} - \epsilon \frac{\partial \mathcal{E}}{\partial d_2}|_{\boldsymbol{d}=\widehat{\boldsymbol{d}}^{(k)}},$$

with initial value supplied by the final value at the previously scanned pixel,

$$\widehat{\boldsymbol{d}}^{(0)}(n_1, n_2) = \widehat{\boldsymbol{d}}^{(\text{final})}(n_1 - 1, n_2).$$

A key reference is [11]; see also [6]. The method works well with just a few iterations when the motion is small, but often fails to converge when the displacements are large. In [12], this differential displacement approach was extended to hierarchically structured motion estimation, with application to image sequence frame interpolation.

Optical Flow Methods

Optical flow is a differential method that works by approximating the derivatives rather than the function error itself, as in block matching. It is a least-squares approximation to the *spatiotemporal constraint equation*,

$$v_x \frac{\partial f}{\partial x} + v_y \frac{\partial f}{\partial y} + \frac{\partial f}{\partial t} = 0, \qquad (11.2\text{–}4)$$

which is derived by partial differentiation of the optical flow equation (11.2–1), rewritten as a function of real variables (x, y) with velocity parameters v_x and v_y,

$$f(x, y, t) = f(x - v_x dx, y - v_y dy, t - \Delta t).$$

Because of noise in the frame, (11.2–4) is then subjected to least squares approximation to give the optical flow velocity estimate. Specifically, we form the error criteria

$$\mathcal{E}_{MV} \triangleq \left(v_x \frac{\partial f}{\partial x} + v_y \frac{\partial f}{\partial y} + \frac{\partial f}{\partial t} \right)^2,$$

to be minimized over local regions.

In practice, a smoothing term must be added to this error term to *regularize* the estimate, which otherwise would be much too rough—i.e., too much high-frequency energy in the estimate $\widehat{\boldsymbol{v}}(x, y, t)$. In the Horn and Schunck method [13], a gradient smoothness term is introduced via a Lagrange multiplier as

$$\begin{aligned} \lambda \mathcal{E}_S &\triangleq \lambda \left[\|\nabla v_x\|^2 + \|\nabla v_y\|^2 \right] \\ &= \lambda \left[\left(\frac{\partial v_x}{\partial x} \right)^2 + \left(\frac{\partial v_x}{\partial y} \right)^2 + \left(\frac{\partial v_y}{\partial x} \right)^2 + \left(\frac{\partial v_y}{\partial x} \right)^2 \right], \end{aligned}$$

which, for large values of the positive parameter λ, makes the velocity estimate change slowly as a function of the spatial variables x and y.

Integrating over the area of the image, we get the total error to be minimized as

$$\begin{aligned}\mathcal{E}_T &= \int\int (\mathcal{E}_{MV} + \lambda\mathcal{E}_S)\, dxdy \\ &= \int\int \left\{ \left(v_x \frac{\partial f}{\partial x} + v_y \frac{\partial f}{\partial y} + \frac{\partial f}{\partial t} \right)^2 + \lambda \left[\left(\frac{\partial v_x}{\partial x} \right)^2 + \left(\frac{\partial v_x}{\partial y} \right)^2 \right.\right. \\ &\quad \left.\left. + \left(\frac{\partial v_y}{\partial x} \right)^2 + \left(\frac{\partial v_y}{\partial x} \right)^2 \right] \right\} dxdy.\end{aligned}$$

We seek the minimizing velocity vector

$$\widehat{\boldsymbol{v}} \triangleq \arg\min \mathcal{E}_T(\boldsymbol{v}).$$

The calculus of variations is then used to find the minimum of this integral in terms of the unknown functions $v_x(x,y,t)$ and $v_y(x,y,t)$ for each fixed frame t. The resulting equations are then approximated using first-order approximations for the various derivatives involved. Longer digital filters may provide improved estimates of these derivatives of the, assumed bandlimited, analog image frames [14]. An iterative solution is then obtained using Gauss-Seidel iterations.

While this estimate has been used extensively in computer vision, it is not often used in video compression because of its rather dense velocity estimate. However, optical flow estimates have been used extensively in video filtering, where the need to transmit the resulting motion vectors does not occur. There it can give a smooth and consistent performance, with few motion artifacts. A modern optical flow method is presented in Section 5.4 of Chapter 5 in *The Essential Guide to Video* [15]. The main problem with optical flow methods is that the smoothness of their motion does not allow discontinuities of motion across object boundaries in the scene.

Mesh-Based Methods

In a mesh-based method, similar to block-based, a regular grid of velocity points called *control points* is set up in the target frame and the corresponding matched points in a reference frame. But unlike block matching, the motion is not considered constant between the control points. Rather, these points are used to set up an *affine motion model*. An affine model has six parameters and represents rotation and translation, projected onto the image plane, as

$$\begin{aligned} d_1(x_1,x_2) &= a_{11}x_1 + a_{12}x_2 + a_{13}, \\ d_2(x_1,x_2) &= a_{21}x_1 + a_{22}x_2 + a_{23}, \end{aligned}$$

where the position vector $\boldsymbol{x} = (x_1,x_2)^T$ and $\boldsymbol{d}(\boldsymbol{x})$ is the displacement vector. We can see translational motion as the special case where only a_{13} and a_{23} are nonzero and the displacement is constant at $(d_1,d_2) = (a_{13},a_{23})$ in this block. The motion warping

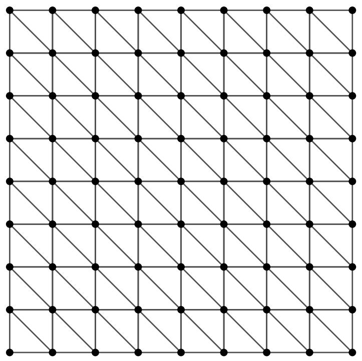

FIGURE 11.2–7

Illustration of regular triangular mesh grid on target frame.

effect of an affine motion model has been found to well approximate the apparent motion or optical flow of pixels on rigid objects. If we break up the squares of the regular grid in the target frame into triangles, we get triangular patches, with three control points at the vertices of each triangle, as seen in Figure 11.2–7, and a separate affine model can be determined for each triangle patch—i.e., six linear equations in the six unknowns $(a_{11}, a_{21}, a_{12}, a_{22}, a_{13}, a_{23})$.

Because the control points are shared between two adjoining patches, the resulting velocity field will be continuous across the patch boundaries unlike the case in block matching. In the reference frame, when the control points are properly matched, the triangular grid will appear warped. This grid shape indicates that spatially warped prediction is being applied from the reference frame to the target frame. The continuity of the velocity field makes the prediction more subjectively acceptable but does not usually lead to better objective error performance in either an MSE or MAE sense. Basically, a small geometric error is hardly noticeable, but it can affect the objective error a lot. Pseudo code for performing mesh matching follows:

```
FOR each grid point
    Do block matching, with block centered by grid point, to find $\boldsymbol{d}_i$.
    $T_i = 1$;
END FOR
WHILE not exceed maximum iterations
    FOR each grid point $\boldsymbol{V}_i$
            IF $T_i == 1$
                 Refine the motion vector by $\min_{\boldsymbol{d}_i} \sum_k E_k$, where $E_k$ is
                 the prediction error of each triangle that connected
                 to this grid point
            IF $\boldsymbol{d}_i$ does not change
                 $T_i = 0$;
            ELSE
                 $T_i = 1$;
```

```
                FOR any grid point $V_j$ that edge-connects with $V_i$
                $T_j = 1$;
                END FOR
    END FOR
    IF $\sum T_i < 1$ BREAK
END WHILE
```

This algorithm first performs block matching to get initial displacement estimates for the control grid points. It then iteratively visits all the grid points in succession, looking for better affine model fits. The T_i variable keeps track of whether the control vector at that grid point is converged or not. The following example has used this algorithm with the MAE error criteria to perform a mesh-based motion estimation. The max number of iterations was set to three, and the search area was set at $(\pm 31, \pm 31)$.

Example 11.2–1: Mesh Matching Versus Block Matching

We look at two examples of warping frame 93 of the CIF[5] clip *foreman* at 30 fps to match the next frame 94. There is enough motion between these two frames to illustrate the shortcomings of each approach. We use fixed-size 32 × 32 blocks here. Figure 11.2–8 shows frame 94 with the fixed-size triangular grid overlaid upon it. Figure 11.2–9 shows frame 93 with the found warped triangular mesh overlaid. We see that there is considerable movement between these two frames, and it is mainly in the foreman's mouth and chin region. Figure 11.2–10 shows the resulting spatially warped estimate of frame 94. It clearly displays warping errors, most obvious in the lower facial region. Finally, in

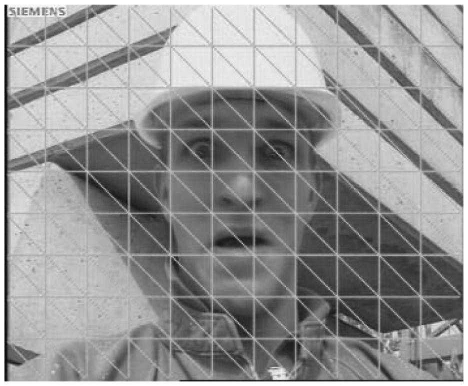

FIGURE 11.2–8

Frame 94 of the foreman clip with fixed-size triangular grid overlaid.

[5]See appendix on video formats at the end of this chapter.

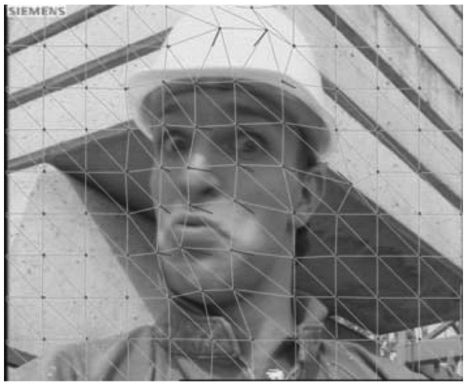

FIGURE 11.2–9

Frame 93 of the foreman clip with warped grid overlaid.

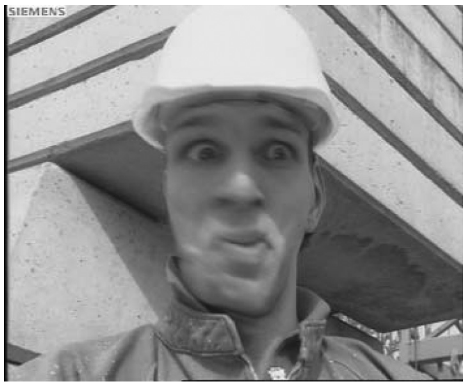

FIGURE 11.2–10

Warped prediction of frame 94 of the foreman clip from the preceding frame using triangular fixed-size mesh matching (see color insert).

Figure 11.2–11 we show the corresponding block-matching estimate for the same 32 × 32 fixed size grid. We can see obvious blocking artifacts here. We see evident distortions in each prediction of frame 94, mostly near the mouth region of the foreman's face. You can watch the full videos included in a folder on this book's Web site. ■

The mesh may be fixed size or variable size, and the motion parameters a_{ij} may be estimated hierarchically or not. A popular choice is *variable-size mesh matching* (VSMM). Use of variable size blocks can reduce both the blocking and warping artifacts in this example. Finally, and distinct from mesh matching, there also exist generalized block models with affine or more general polynomial motion models within each block [16].

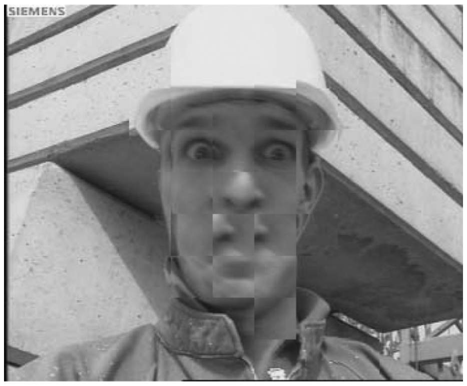

FIGURE 11.2–11

Fixed-size block-matching prediction of frame 94 of the foreman clip from the preceding frame.

11.3 MOTION-COMPENSATED FILTERING

If the motion is slow or there is no motion at all, then simple temporal filtering can be very effective for estimation, restoration, frame-rate change, interpolation, or pre- or postprocessing. In the presence of strong motion, however, artifacts begin to appear in the simple temporal filter outputs and the needed coherence in the input signal begins to break down. Such coherence is needed to distinguish the signal from the noise, distortion, interference, and artifacts that may also be present. A solution is to modify the filter trajectory so that it follows along the trajectory of motion. In this way, signal coherence is maintained, even with moderate to fast motion.

The basic idea is to modify an LSI filter as follows:

$$\begin{aligned} y(n_1,n_2,n) = & \sum_{k_1,k_2} h(k_1,k_2,0)x(n_1-k_1,n_2-k_2,n) \\ & + \sum_{k_1,k_2} h(k_1,k_2,1)x(n_1-d_1-k_1,n_2-d_2-k_2,n-1) \qquad (11.3\text{–}1) \\ & + \text{ etc.} \end{aligned}$$

Here, $d_1 = d_1(n_1,n_2,n)$ is the horizontal component of the displacement vector between frames n and $n-1$, and d_2 is the vertical component of displacement. In order to get the corresponding terms for frame $n-2$, we must add the displacement vectors from the frame pair $n-1$ and $n-2$ to get the correct displacement. We should add them vectorially:

$$d_1'(n_1,n_2) = d_1(n_1,n_2,n) + d_1\big(n_1 - d_1(n_1,n_2,n), n_2 - d_2(n_1,n_2,n), n-1\big),$$
$$d_2'(n_1,n_2) = d_2(n_1,n_2,n) + d_2\big(n_1 - d_1(n_1,n_2,n), n_2 - d_2(n_1,n_2,n), n-1\big).$$

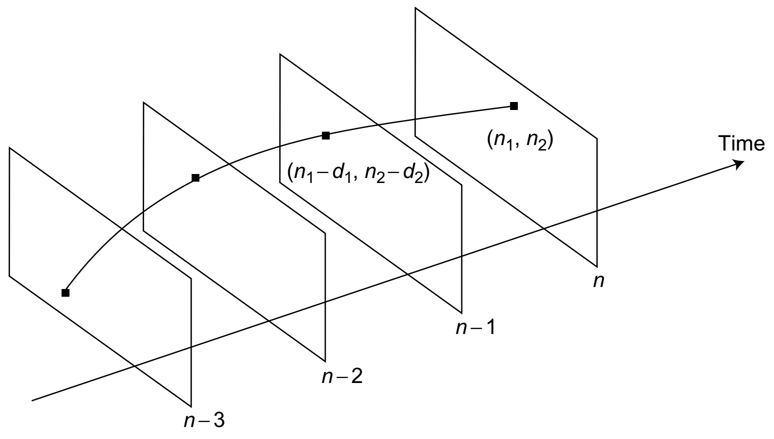

FIGURE 11.3–1

An illustration of motion-compensated filtering along a motion path.

Here, we assume that the displacement vectors are known, most likely because they were estimated previously, and that they are integer valued. If they are not integer valued, which is most of the time, then the corresponding signal value, such as $x(n_1 - d_1 - k_1, n_2 - d_2 - k_2, n - 1)$, must itself be estimated via interpolation. Most often, spatial interpolation is used based on the use of various lowpass filters. Effectively, in (11.3–1) we are filtering along the motion paths rather than simply filtering straight forward (or backward) in time (n) at each spatial location. One way to conceptualize this is via the diagram in Figure 11.3–1.

MC-Wiener Filter

In Figure 11.3–2, MC denotes *motion-compensated warping* performed on the noisy observations $y(n_1, n_2, n)$. The Wiener filtering is then done on the warped data in the MC domain, with signal and noise PSDs calculated from some similar MC data. Finally, the *inverse motion compensation* (IMC) operator dewarps the frames back to original shape to produce the output estimate $\widehat{x}(n_1, n_2, n)$. Three-dimensional MC-Wiener filtering was introduced in [17]. The concept of IMC depends on the motion field being one-to-one. In a real image sequence, there is a relatively small number of pixels where this is not true due to coverings and uncoverings of objects in the scene, the so-called *occlusion problem*. In these areas, some approximation must be used in Figure 11.3–2 in order to avoid introducing artifacts into the final video estimate. It is common to resort to intraframe filtering in these occluded areas.

Because of the strong correlation in the temporal direction in most video, there is often a lot to gain by processing the video jointly in both the temporal and spatial directions. Since a Wiener filter is usually implemented as an FIR filter, exploitingthe

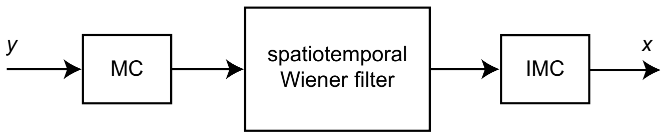

FIGURE 11.3–2

Illustration of MC warping followed by Wiener filter followed by IMC warping (IMC).

temporal direction in this way means that several to many frames must be kept in active memory. An alternative to this method is the spatiotemporal Kalman filter, which can use just one frame of memory to perform its recursive estimate, a motion-compensated version of which is presented next. Of course, both methods require the estimation of a suitable image sequence model and noise model. The signal model spectra can be obtained via estimation on similar noise-free data for the Wiener filter, while the Kalman filter needs parameter estimation of an autoregressive model. Both models must be trained or estimated on data that have been warped by the motion compensator, since this warped domain is where their estimate is performed.

MC-Kalman Filter

The basic idea here is that we can apply the totally ordered temporal 3-D RUKF of Chapter 10 along the motion trajectory using a good motion estimator that approximates true motion. As before, we can use multiple models for both motion and image estimation. To reduce object blurring and sometimes even double images, we effectively shift the temporal axis of the filter to be aligned with motion trajectories. When a moderate-sized moving object is so aligned, we can then apply the filter along the object's trajectory of motion by filtering the MC video. Since the MC video has a strong temporal correlation, its image sequence model will have a small prediction error variance. This suggests that high spatial frequencies can be retained even at low input SNRs via motion-compensated Kalman filtering. The overall block diagram of a motion compensated 3-D Kalman filter of Woods and Kim [1], or MC-RUKF, is shown in Figure 11.3–3.

This motion-compensated spatiotemporal filter consists of three major parts: the motion estimator, the motion compensator, and the 3-D RUKF. While filtering a video, two different previous frames could be used for motion estimation; one could use either the previous smoothed frame $E\{\boldsymbol{x}(n-1)|\boldsymbol{y}(n),\boldsymbol{y}(n-1),\ldots\}$ or the previous noisy frame $\boldsymbol{y}(n-1)$. In our work, we have generally found it best to use the smoothed previous frame, since it is the best estimate currently available. For motion estimation, we used an HBM method.

The motion estimate is used to align a set of local frames along the motion trajectory. To effect this local alignment, the smoothed previous frame estimate is displaced to align with the current frame. In an iterative method extension shown in Figure 11.3–3, two smoothed frames are used to improve on the initial motion estimates. These smoothed frames retain spatial high frequencies and have reduced

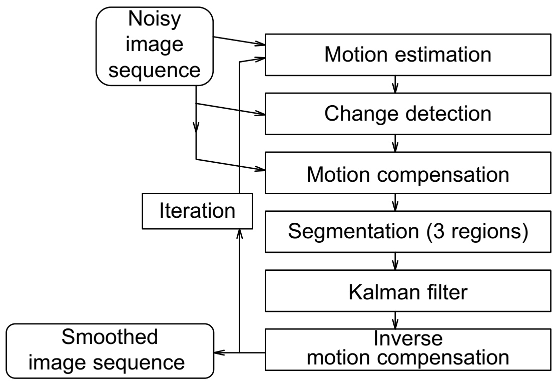

FIGURE 11.3–3

System diagram for motion-compensated spatiotemporal Kalman filter.

noise, so that these frames can now be used for motion estimation with a smaller size block. A motion vector field is then estimated from these two frames (i.e., $E\{\boldsymbol{x}(n-1)|\boldsymbol{y}(n),\boldsymbol{y}(n-1),\ldots\}$ and $E\{\boldsymbol{x}(n)|\boldsymbol{y}(n+1),\boldsymbol{y}(n),\boldsymbol{y}(n-1),\ldots\}$), followed by a second application of the steady-state 3-D RUKF. A few iterations suffice.

Multimodel MC-RUKF

The local correlation between two MC frames depends on the accuracy of the motion estimation. Since the SNR of the noisy observed video will be low, the motion estimation has a further limitation on its accuracy, due to statistical variations. To deal with the resulting limited motion vector accuracy, we can use a variable number of models to match the various motion regions in the image; for example, we can use three motion models: *still*, *predictable*, and *unpredictable*. The motivation here is that in the still region, we can perform unlimited temporal smoothing at each pixel location. In the predictable region, there is motion, but it is motion that can be tracked well by our motion estimator. Here, we can smooth along the found motion trajectory with confidence. Finally, in the unpredictable region, we find that our motion estimate is unreliable and so fall back on the spatial RUKF there. This *multiple model* version (MM MC-RUKF) results in a very high temporal coherence in the still region, high temporal coherence in the predictable region, and no motion blurring in the unpredictable region. The segmentation is based on local variance of the *displaced frame difference* (DFD).

As mentioned earlier, we employ a block-matching method for motion estimation. Even when there is no correspondence between two motion-compensated frames, the block-matching method chooses a pixel in the search area that minimizes the displaced frame difference measure. However, the estimate will probably not have much to do with the real motion, and this can lead to low temporal correlation in the unpredictable region. This is the so-called *noisy motion vectors* problem. We can compensate for this, in the case of still regions, by detecting them with an extra step, based on the frame difference. We filter the frame difference and use a simple 7×7 box filter to reduce the effect of the observation noise. Also, a 3×3 box filter is used on the MC output to detect the predictable region. The outputs are then fed into local variance detectors. We found that when a pixel in a still region was miss-detected as in the predictable region, a visual error in the filtered image sequence was noticeable, while in the opposite case, the error was not noticeable. Hence, we detect the still region again in the filtering step. Three spatiotemporal AR models are obtained from the residual video of the original sequence for our simulation. For more details, see [1].

Example 11.3–1: MM MC-RUKF Experimental Result

We used the CIF video salesman, which is monochrome and of size 360×280 pixels at 15 fps. We then added white Gaussian noise to achieve a 10-dB SNR. The processing parameters of the MC-RUKF were as follows: image model order $1 \times 1 \times 1$, update region $2 \times 2 \times 1$, and final MC block sizes of both 9×9 and 5×5. The 3-D AR model obtained from the original (modified) video was used. This model could also have been obtained from the noisy video or from a prototype noise-free, with some loss of performance. (Based on existing work in the identification of 2-D image models, it is our feeling that the additional loss would not be great.) We used a steady-state gain array, calculated off-line on a small fictitious image sequence. The SNR improvement achieved was 6–8 dB with the 3-D RUKF alone, with an additional MC-RUKF improvement of about 1 dB. Using the multimodel feature, a further MM MC-RUKF improvement of about 1 dB was achieved, totaling to an 8- to 10-dB improvement or an output SNR of 18–20 dB.

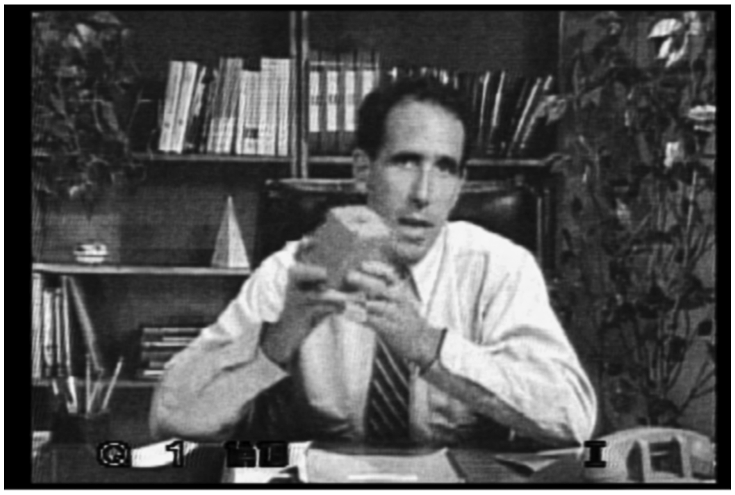

FIGURE 11.3–4

A frame from the MC-RUKF.

The restored video in Figures 11.3–4 and 11.3–5 showed motion artifacts visible in some motion areas but was generally quite visually pleasing.

The resulting SNR improvement curves are given in Figure 11.3–6. We notice that the MM MC-RUKF provides the best objective performance by this MSE-based measure. We can see an initial start-up transient of about 10 frames. We notice also how the up

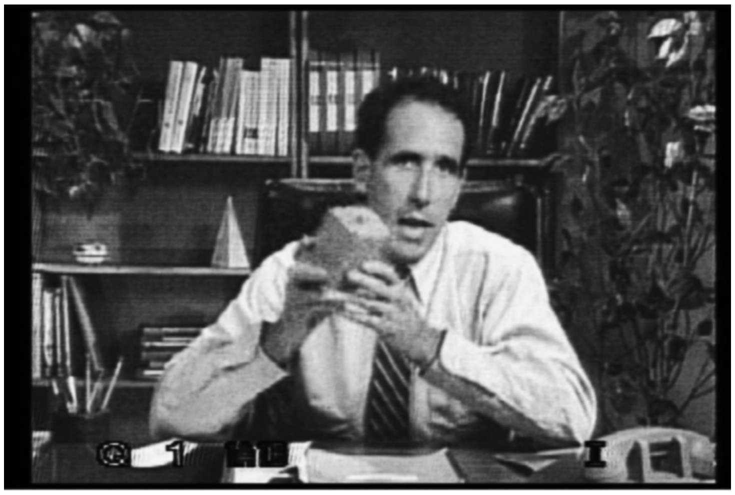

FIGURE 11.3–5

A frame from the MM MC-RUKF.

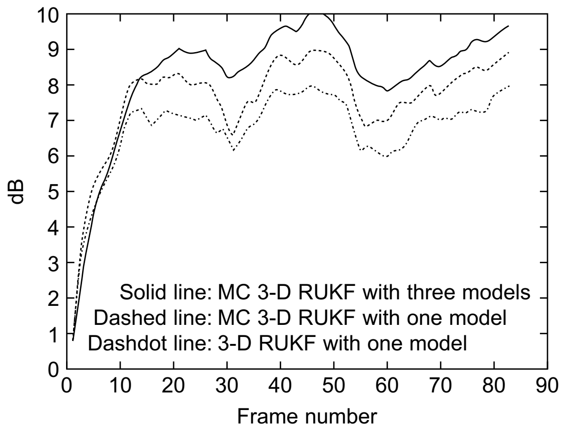

FIGURE 11.3–6

Plot of SNR improvements versus frame number for the MM MC-RUKF (3 models), MC-RUKF, and 3-D RUKF on the noisy salesman clip.

to 10-dB improvement varies across the frame number; this is caused by the motion of objects and moving shadows in the scene. Videos are available for download at the book's Web site.

Frame-Rate Conversion

Frame-rate conversion is needed today due to the coexistence of multiple standard frame rates (60, 30, 25, and 24 fps), and also leads to a convenient separation of acquisition format, transmission standard, and viewing or display format. Frame-rate up-conversion is also often used to double (or quadruple) the frame rate for display (e.g., 25 fps to 50 or 100 fps). There are various methods for increasing the frame rate, the simplest being frame repeat (i.e., sample and hold-in time). Somewhat more complicated is making use of a straight temporal average, without motion compensation. The filtering is a 1-D interpolation (i.e., linear filtering) in the temporal direction, done for each pixel separately. A potentially more accurate method for frame-rate increase is MC-based frame interpolation, first suggested in [8].

Example 11.3–2: Frame-rate Up Conversion

We have applied the method of [8] to the test clip Miss America, which is color and SIF sized[6], with 150 frames at 30 fps. To perform our simulation, we first decimated it down to 5 fps and used only this low frame rate as input. Then we used the following strategies to interpolate the missing frames (i.e., raise the frame rate back up to 30 fps): frame replication, linear averaging, and motion-compensated interpolation. As a motion estimation method, we employed HBM, with the result smoothed by a simple lowpass filter.

Figure 11.3–7 shows a frame from temporal up-conversion using a simple linear averaging filter. Note that during this time period, there was motion that has caused a double image effect on the up-converted frame. Figure 11.3–8 shows the result of using the motion-compensated up-conversion at a frame number near to that of the linear result in Figure 11.3–7. We do not see any double image, and the up-conversion result is generally artifact-free.

In this case our translational motion model worked very well, in part because of the rather simple motion displayed in this Miss America test clip. However, it does not always work this well, and MC up-conversion remains a challenging problem. Videos are available for download at the book's Web site.

Deinterlacing

As mentioned in Example 2.2–5 of Chapter 2, deinterlacing is used to convert from a conventional interlaced format to one that is progressive or noninterlaced. In so

[6]Please see appendix on video formats.

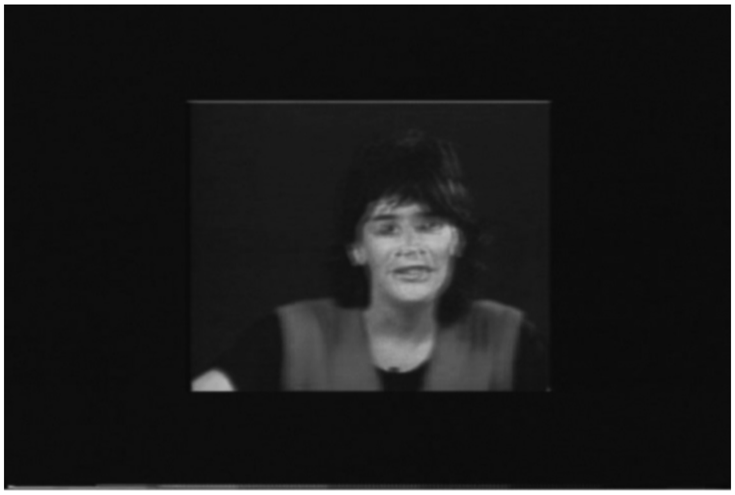

FIGURE 11.3–7

A frame from a linearly interpolated temporal up-conversion of the Miss America clip from 5 to 30 fps.

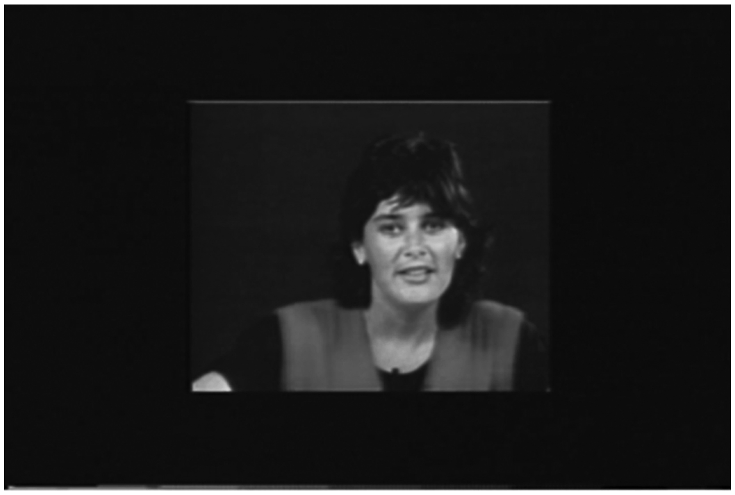

FIGURE 11.3–8

A frame from the motion-compensated temporal up-conversion of the Miss America clip from 5 to 30 fps.

doing, the deinterlacer must estimate the missing data—i.e., the odd lines in the so-called *even frames* and the even lines in the *odd frames*. A conventional deinterlacer uses a diamond *v–t* multidimensional filter in upsampling the data to progressive format. If the interlaced video had been prefiltered prior to its original sampling on this lattice to avoid spatial frequency aliasing, then using an ideal filter, the progressive reconstruction can be exact, but still with the original spatiotemporal frequency response. If proper prefiltering had not been done at the original interlaced sampling, then aliasing error may be present in both the interlaced and progressive data. In this case, a nonideal frequency response for the conversion filter can help to suppress the alias energy that usually occurs at locations of high spatiotemporal frequency. While interlaced video is not as ubiquitous as it once was, the ATSC broadcast interlaced standard 1080i is common and needs conversion to 1080p for a progressive display.

Example 11.3–3: Conventional Deinterlacer

This example uses a 9×9 diamond-shaped support in the $v \times t$ plane. The filter coefficients, shown in Table 11.3–1, were obtained via window-based FIR filter design. The frequency response of this filter is shown in Figure 11.3–9, where we can see a broader response along both temporal and vertical frequency axes than along diagonals, hence approximating a diamond pattern in the $v \times t$ $(n_2 \times n)$ frequency domain.

Figure 11.3–10 shows one field from the interlaced salesman sample, obtained by filtering and downsampling from the corresponding progressive clip. It serves as the starting point for our deinterlacing experiments. Figure 11.3–11 shows a frame from the resulting *progressive* (noninterlaced) output. We note that the result is generally pleasing, if somewhat soft (slightly blurred). From the frequency response in Figure 11.3–9, we can see that the image frame sharpness should be generally preserved for low temporal frequencies (i.e., slowly moving or stationary objects). Fast-moving objects, corresponding to diagonal support on the $v \times t$ filter frequency response function, will be blurred. Videos are available for download at the book's Web site. ■

While blurring of fast-moving objects is generally consistent with the limitations of the human visual system response function, coherent motion can be tracked by the viewer. As such, it appears as low temporal frequency on the tracking viewer's retina, and hence the blurring of medium- to fast-moving objects can be detected for so-called *trackable motion*.

Table 11.3–1 Diamond Filter Coefficients

0	0	0	0	0.001247	0	0	0	0
0	0	0	0.004988	−0.005339	0.004988	0	0	0
0	0	0.007481	−0.016016	−0.013060	−0.016016	0.007481	0	0
0	0.004988	−0.016016	−0.036095	0.162371	−0.036095	−0.016016	0.004988	0
0.001247	−0.005339	−0.013060	0.162371	0.621808	0.162371	−0.013060	−0.005339	0.001247
0	0.004988	−0.016016	−0.036095	0.162371	−0.036095	−0.016016	0.004988	0
0	0	0.007481	−0.016016	−0.013060	−0.016016	0.007481	0	0
0	0	0	0.004988	−0.005339	0.004988	0	0	0
0	0	0	0	0.001247	0	0	0	0

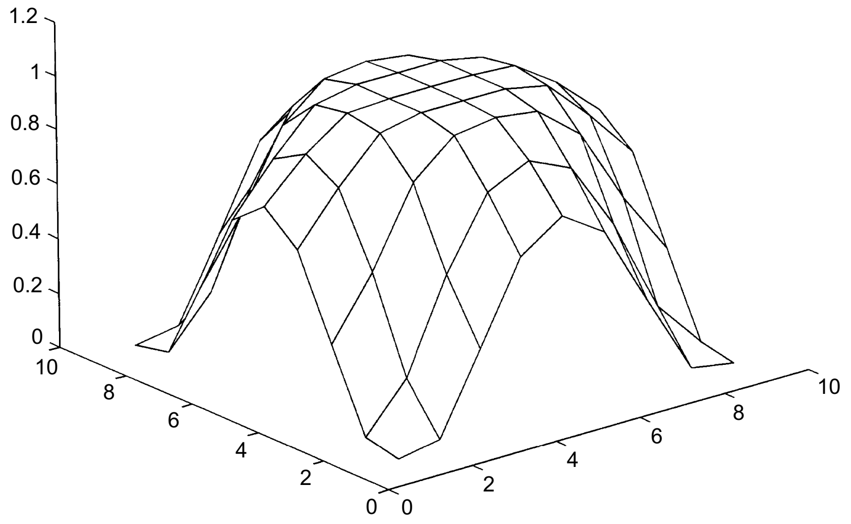

FIGURE 11.3–9

Sketch of diamond filter response in the vertical–temporal frequency domain.

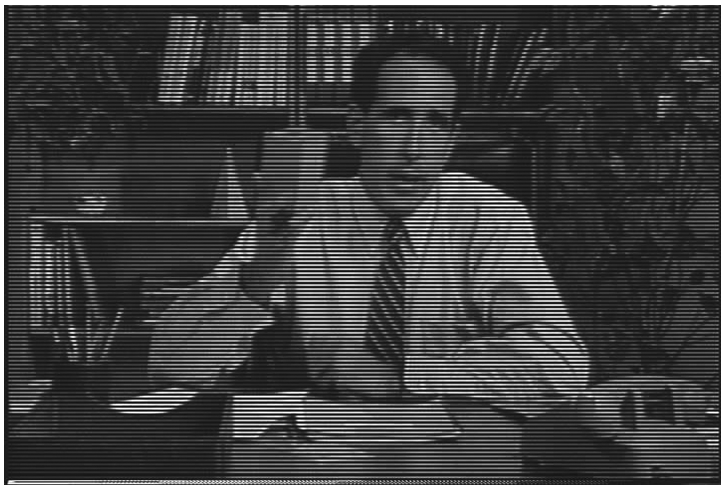

FIGURE 11.3–10

One field from the interlaced version of the salesman clip.

Example 11.3–4: Median Deinterlacer

An alternative to the use of the classic multdimensional filter is the vertical–temporal median filter. The most common method uses a three-pixel median filter, with one pixel

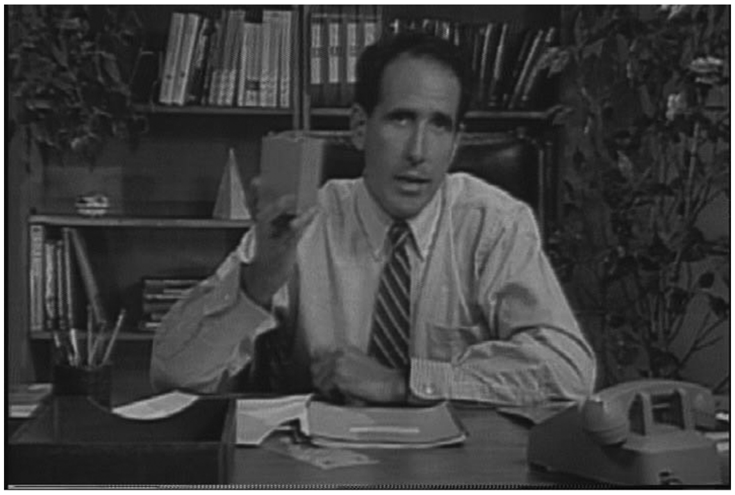

FIGURE 11.3–11

A progressive frame from the diamond filter $(v \times t)$ output for an interlaced input.

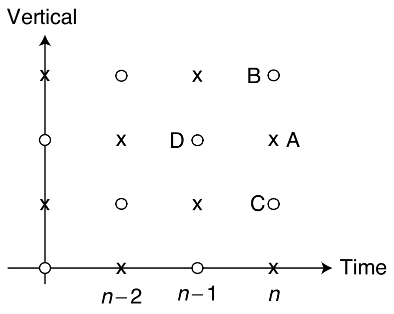

FIGURE 11.3–12

Illustration of pixels input (B, C, and D) to the median filter deinterlacer.

above and below the current pixel, and one pixel right behind it in the previous field. On the progressive lattice, this median operation can be written as follows:

for odd frames,

$$\widehat{x}(n_1, 2n_2, n) = \text{median}\{x(n_1, 2n_2 + 1, n), x(n_1, 2n_2 - 1, n), x(n_1, 2n_2, n - 1)\},$$

for even frames,

$$\widehat{x}(n_1, 2n_2 + 1, n) = \text{median}\{x(n_1, 2(n_2 + 1), n), x(n_1, 2n_2, n), x(n_1, 2n_2 + 1, n - 1)\}.$$

In Figure 11.3–12, circles indicate pixels (lines) present in a field, while $\times$'s represent missing pixels (lines). We see three input pixels (B, C, and D) and one output

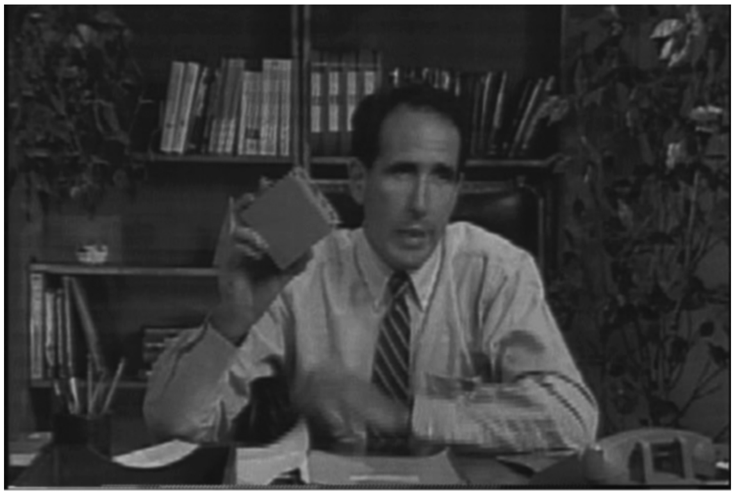

FIGURE 11.3–13

A deinterlaced frame of the salesman clip by the adaptive median filter.

pixel (A), which represent a missing pixel at field n. The statistical median of the three input values (B, C, and D) will tend to favor D if there is no motion, but to favor B or C in the case of motion. Thus this simple median deinterlacer switches back and forth between temporal and spatial (vertical) interpolation to fill in the missing pixel lines in the even and odd fields. A vertical–temporal median deinterlaced frame is shown in Figure 11.3–13. While the result is sharper than that of the multidimensional filter, this sharpness is obtained at the cost of small artifacts occurring on fast-moving objects. Videos are available for download at the book's Web site. ■

A more powerful alternative is motion-compensated deinterlacing. It uses motion estimation to find the best pixels in the previous field or fields for the prediction of the current pixel in the missing lines of the current field.

Example 11.3–5: Motion-Compensated Deinterlacer

In this example we try to detect and track motion [18] and then use it to perform the deinterlacing. Using an HBM motion estimator based on a QMF SWT, we first determine if the velocity is zero or not, based on looking at a local average of the mean-square frame difference and comparing it to a threshold. A simple three-tap vertical interpolation filter was used to deinterlace at this first stage. The motion is then said to be "trackable" if the local motion-compensated MSE is below a second threshold. The algorithm then proceeds as follows:

- When no motion is detected, we smooth in the temporal direction only (i.e., use the pixel at the same position in the previous field).

- When motion is detected, and with reference to Figure 11.3–14, we project the motion path onto the previous two fields, with a *cone of uncertainty* opening to 0.1–0.2 pixels at the just prior field. If the cone includes a pixel in the first prior field, then that pixel is copied to the missing pixel location A in the current field. Otherwise, we look to the second prior field. If no such previous pixels exist in the "cone regions," we perform linear spatiotemporal interpolation.

The result in Figure 11.3–15 is potentially the best of these examples shown, albeit the most computationally demanding due to the motion estimation. It is sharp and clear, but unfortunately suffers from some motion artifacts, which could at least be partially ameliorated by more sophisticated motion estimation and compensation methods. The frame rate of the salesman clip was 15 fps.

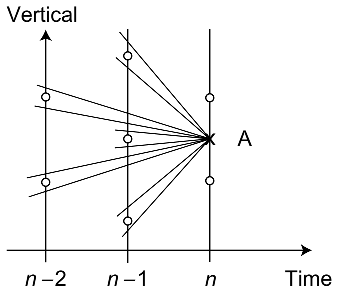

FIGURE 11.3–14

An illustration of the cone approach to motion-compensated deinterlacing.

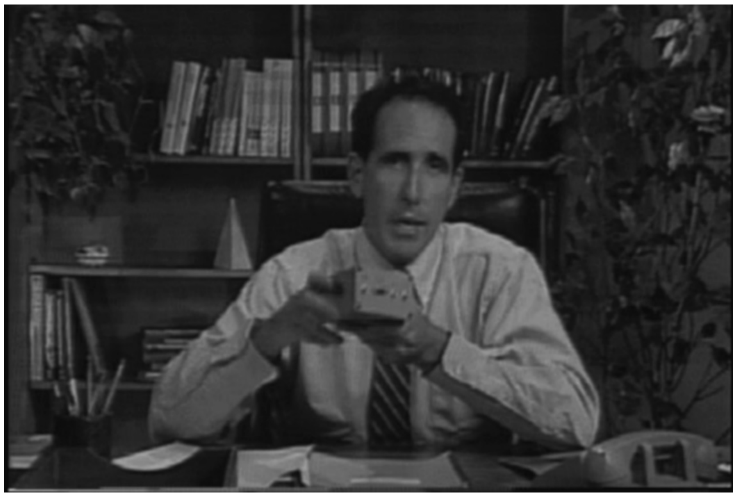

FIGURE 11.3–15

An MC deinterlaced frame from the salesman clip.

One last comment on deinterlacing: Of course, missing data not sampled cannot be recovered without some assumptions on the original continuous-parameter data. In the specific case of deinterlacing, and in the worst case, one can imagine a small feature one pixel high, moving at certain critical velocities, such that it is either always or never present on the interlaced grid. In real life, the velocity would not exactly match a critical velocity and so the feature would appear and disappear at a nonzero temporal frequency. If this feature is the edge of a rectangle or part of a straight line, the flickering can be noticed by the eye, but may be very hard to correct. Therefore, an issue in MC deinterlacers is the consistency of the estimate. A recursive block estimate showing a high degree of visual consistency is given in de Haan et al. [19].

11.4 BAYESIAN METHOD FOR ESTIMATING MOTION

In Chapter 8 we introduced Bayesian methods for image estimation and restoration, which use a Gibbs-Markov signal model together with some, most often iterative, solution method, such as simulated annealing (SA). Other methods used include deterministic iterative methods such as *iterated conditional mode* (ICM) and *mean field annealing* (MFA). ICM is a method that sweeps through the sites (pixels) and maximizes the conditional probability of each pixel in sequence. It is fast but tends to get stuck at local optima of the joint conditional probability, rather than find the global maxima. MFA is an annealing technique that assumes that the effect of a neighboring clique's potential function can be well modeled with its mean value. While this changes the detailed global energy function, the iteration proceeds much more quickly and reportedly provides very nice results, generally classified as being somewhere between ICM and SA. A general review of these approaches is contained in the review article by Stiller and Konrad [20].

To extend the Gibbs-Markov model, we need to model the displacement vector or motion field $\boldsymbol{d}$, which can be done as

$$f_{\boldsymbol{d}}(\boldsymbol{D}) = K \exp[-U_{\boldsymbol{d}}(\boldsymbol{D})],$$

where the matrix $\boldsymbol{D}$ contains the values of the vector field $\boldsymbol{d}$ on the image region or frame. The energy function $U_{\boldsymbol{d}}(\boldsymbol{D})$ is the sum of potential function over all the pixels (sites) $\boldsymbol{n}$ and their corresponding cliques,

$$U_{\boldsymbol{d}}(\boldsymbol{D}) \triangleq \sum_{\boldsymbol{n}} \sum_{c_n \in \mathcal{C}} V_{c_{\boldsymbol{n}}}(\boldsymbol{D}),$$

where $\boldsymbol{C_n}$ denotes the clique system for the displacement field $\boldsymbol{d}$, over frame $\boldsymbol{n}$.

A common setup calls for the estimation of the displacement between two frames, $\boldsymbol{X}_n$ and $\boldsymbol{X}_{n-1}$, and using the MAP approach, we seek the estimate

$$\widehat{\boldsymbol{D}} = \arg\max_{\boldsymbol{D}} f(\boldsymbol{D}|\boldsymbol{X}_n, \boldsymbol{X}_{n-1}). \qquad (11.4\text{–}1)$$

This simple model can be combined with a line field on an interpixel grid, as in Chapter 8, to allow for smooth estimates of displacement that respect the sharp