Chapter 8

Dense motion estimation

R. Szeliski, *Computer Vision: Algorithms and Applications*, Texts in Computer Science,
DOI 10.1007/978-1-84882-935-0_8, © Springer-Verlag London Limited 2011

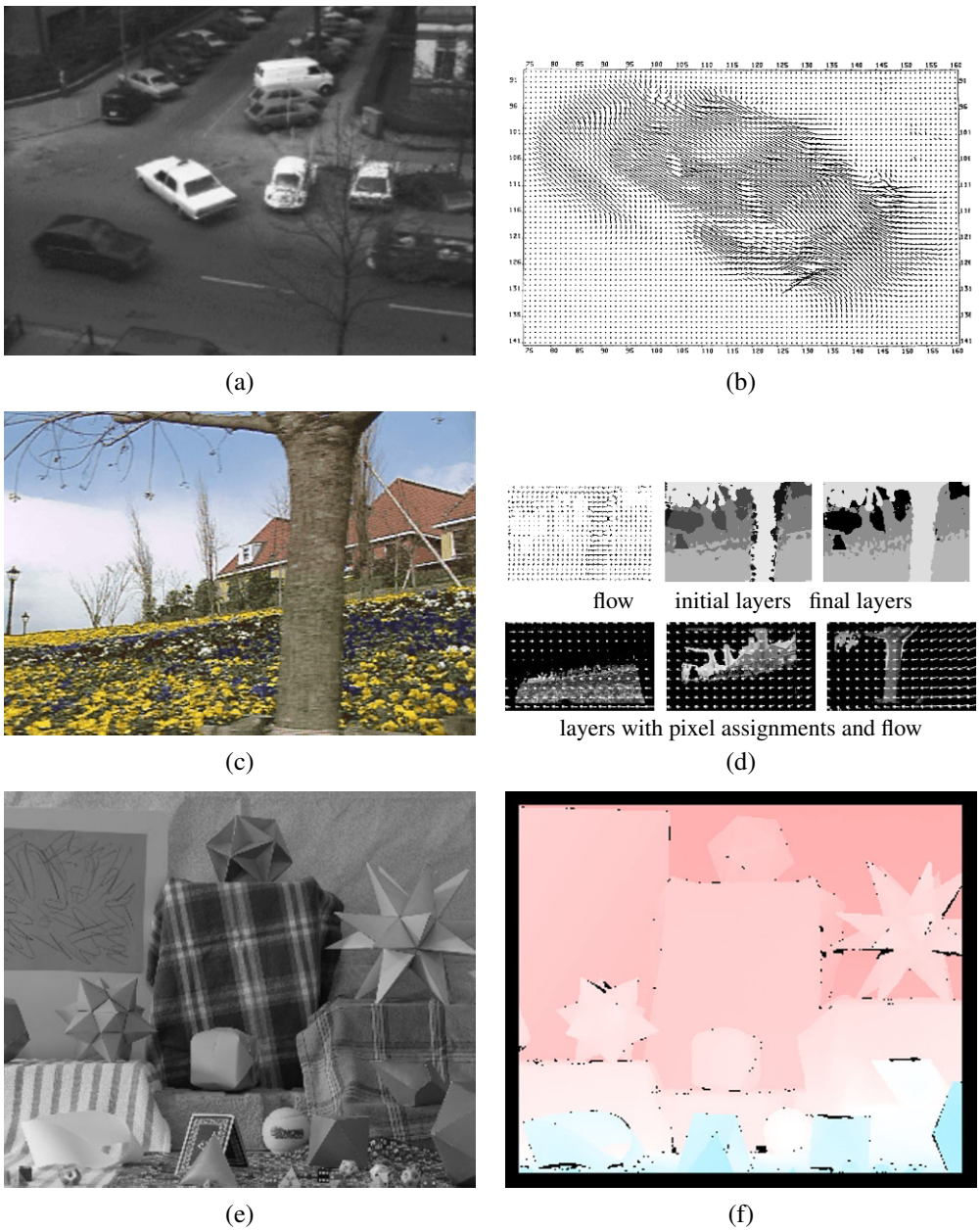


Figure 8.1 Motion estimation: (a–b) regularization-based optical flow (Nagel and Enkelmann 1986) © 1986 IEEE; (c–d) layered motion estimation (Wang and Adelson 1994) © 1994 IEEE; (e–f) sample image and ground truth flow from evaluation database (Baker, Black, Lewis *et al.* 2007) © 2007 IEEE.

Algorithms for aligning images and estimating motion in video sequences are among the most widely used in computer vision. For example, frame-rate image alignment is widely used in camcorders and digital cameras to implement their image stabilization (IS) feature.

An early example of a widely used image registration algorithm is the patch-based translational alignment (optical flow) technique developed by Lucas and Kanade (1981). Variants of this algorithm are used in almost all motion-compensated video compression schemes such as MPEG and H.263 (Le Gall 1991). Similar parametric motion estimation algorithms have found a wide variety of applications, including video summarization (Teodosio and Bender 1993; Irani and Anandan 1998), video stabilization (Hansen, Anandan, Dana *et al.* 1994; Srinivasan, Chellappa, Veeraraghavan *et al.* 2005; Matsushita, Ofek, Ge *et al.* 2006), and video compression (Irani, Hsu, and Anandan 1995; Lee, ge Chen, lung Bruce Lin *et al.* 1997). More sophisticated image registration algorithms have also been developed for medical imaging and remote sensing. Image registration techniques are surveyed by Brown (1992), Zitov'aa and Flusser (2003), Goshtasby (2005), and Szeliski (2006a).

To estimate the motion between two or more images, a suitable *error metric* must first be chosen to compare the images (Section 8.1). Once this has been established, a suitable *search* technique must be devised. The simplest technique is to exhaustively try all possible alignments, i.e., to do a *full search*. In practice, this may be too slow, so *hierarchical* coarse-to-fine techniques (Section 8.1.1) based on image pyramids are normally used. Alternatively, Fourier transforms (Section 8.1.2) can be used to speed up the computation.

To get sub-pixel precision in the alignment, *incremental* methods (Section 8.1.3) based on a Taylor series expansion of the image function are often used. These can also be applied to *parametric motion models* (Section 8.2), which model global image transformations such as rotation or shearing. Motion estimation can be made more reliable by *learning* the typical dynamics or motion statistics of the scenes or objects being tracked, e.g., the natural gait of walking people (Section 8.2.2). For more complex motions, piecewise parametric *spline motion models* (Section 8.3) can be used. In the presence of multiple independent (and perhaps non-rigid) motions, general-purpose *optical flow* (or *optic flow*) techniques need to be used (Section 8.4). For even more complex motions that include a lot of occlusions, *layered motion models* (Section 8.5), which decompose the scene into coherently moving layers, can work well.

In this chapter, we describe each of these techniques in more detail. Additional details can be found in review and comparative evaluation papers on motion estimation (Barron, Fleet, and Beauchemin 1994; Mitiche and Bouthemy 1996; Stiller and Konrad 1999; Szeliski 2006a; Baker, Black, Lewis *et al.* 2007).

8.1 Translational alignment

The simplest way to establish an alignment between two images or image patches is to shift one image relative to the other. Given a *template* image $I_0(\boldsymbol{x})$ sampled at discrete pixel locations $\{\boldsymbol{x}_i = (x_i, y_i)\}$, we wish to find where it is located in image $I_1(\boldsymbol{x})$. A least squares solution to this problem is to find the minimum of the *sum of squared differences* (SSD) function

$$E_{\mathrm{SSD}}(\boldsymbol{u}) = \sum_i [I_1(\boldsymbol{x}_i + \boldsymbol{u}) - I_0(\boldsymbol{x}_i)]^2 = \sum_i e_i^2, \tag{8.1}$$

where $\boldsymbol{u} = (u, v)$ is the *displacement* and $e_i = I_1(\boldsymbol{x}_i + \boldsymbol{u}) - I_0(\boldsymbol{x}_i)$ is called the *residual error* (or the *displaced frame difference* in the video coding literature).[1] (We ignore for the moment the possibility that parts of I_0 may lie outside the boundaries of I_1 or be otherwise not visible.) The assumption that corresponding pixel values remain the same in the two images is often called the *brightness constancy constraint*.[2]

In general, the displacement $\boldsymbol{u}$ can be fractional, so a suitable interpolation function must be applied to image $I_1(\boldsymbol{x})$. In practice, a bilinear interpolant is often used but bicubic interpolation can yield slightly better results (Szeliski and Scharstein 2004). Color images can be processed by summing differences across all three color channels, although it is also possible to first transform the images into a different color space or to only use the luminance (which is often done in video encoders).

Robust error metrics. We can make the above error metric more robust to outliers by replacing the squared error terms with a robust function $\rho(e_i)$ (Huber 1981; Hampel, Ronchetti, Rousseeuw *et al.* 1986; Black and Anandan 1996; Stewart 1999) to obtain

$$E_{\mathrm{SRD}}(\boldsymbol{u}) = \sum_i \rho(I_1(\boldsymbol{x}_i + \boldsymbol{u}) - I_0(\boldsymbol{x}_i)) = \sum_i \rho(e_i). \tag{8.2}$$

The robust norm $\rho(e)$ is a function that grows less quickly than the quadratic penalty associated with least squares. One such function, sometimes used in motion estimation for video coding because of its speed, is the *sum of absolute differences* (SAD) metric[3] or L_1 norm, i.e.,

$$E_{\mathrm{SAD}}(\boldsymbol{u}) = \sum_i |I_1(\boldsymbol{x}_i + \boldsymbol{u}) - I_0(\boldsymbol{x}_i)| = \sum_i |e_i|. \tag{8.3}$$

However, since this function is not differentiable at the origin, it is not well suited to gradient-descent approaches such as the ones presented in Section 8.1.3.

Instead, a smoothly varying function that is quadratic for small values but grows more slowly away from the origin is often used. Black and Rangarajan (1996) discuss a variety of such functions, including the *Geman–McClure* function,

$$\rho_{\mathrm{GM}}(x) = \frac{x^2}{1 + x^2/a^2}, \tag{8.4}$$

where a is a constant that can be thought of as an *outlier threshold*. An appropriate value for the threshold can itself be derived using robust statistics (Huber 1981; Hampel, Ronchetti, Rousseeuw *et al.* 1986; Rousseeuw and Leroy 1987), e.g., by computing the *median absolute deviation*, $MAD = \mathrm{med}_i|e_i|$, and multiplying it by 1.4 to obtain a robust estimate of the standard deviation of the inlier noise process (Stewart 1999).

[1] The usual justification for using least squares is that it is the optimal estimate with respect to Gaussian noise. See the discussion below on robust error metrics as well as Appendix B.3.

[2] Brightness constancy (Horn 1974) is the tendency for objects to maintain their perceived brightness under varying illumination conditions.

[3] In video compression, e.g., the H.264 standard (http://www.itu.int/rec/T-REC-H.264), the sum of absolute transformed differences (SATD), which measures the differences in a frequency transform space, e.g., using a Hadamard transform, is often used since it more accurately predicts quality (Richardson 2003).

Spatially varying weights. The error metrics above ignore that fact that for a given alignment, some of the pixels being compared may lie outside the original image boundaries. Furthermore, we may want to partially or completely downweight the contributions of certain pixels. For example, we may want to selectively "erase" some parts of an image from consideration when stitching a mosaic where unwanted foreground objects have been cut out. For applications such as background stabilization, we may want to downweight the middle part of the image, which often contains independently moving objects being tracked by the camera.

All of these tasks can be accomplished by associating a spatially varying per-pixel weight value with each of the two images being matched. The error metric then becomes the weighted (or *windowed*) SSD function,

$$E_{\mathrm{WSSD}}(\boldsymbol{u}) = \sum_i w_0(\boldsymbol{x}_i) w_1(\boldsymbol{x}_i + \boldsymbol{u}) [I_1(\boldsymbol{x}_i + \boldsymbol{u}) - I_0(\boldsymbol{x}_i)]^2, \tag{8.5}$$

where the weighting functions w_0 and w_1 are zero outside the image boundaries.

If a large range of potential motions is allowed, the above metric can have a bias towards smaller overlap solutions. To counteract this bias, the windowed SSD score can be divided by the overlap area

$$A = \sum_i w_0(\boldsymbol{x}_i) w_1(\boldsymbol{x}_i + \boldsymbol{u}) \tag{8.6}$$

to compute a *per-pixel* (or mean) squared pixel error E_{WSSD}/A. The square root of this quantity is the *root mean square* intensity error

$$RMS = \sqrt{E_{\mathrm{WSSD}}/A} \tag{8.7}$$

often reported in comparative studies.

Bias and gain (exposure differences). Often, the two images being aligned were not taken with the same exposure. A simple model of linear (affine) intensity variation between the two images is the *bias and gain* model,

$$I_1(\boldsymbol{x} + \boldsymbol{u}) = (1 + \alpha) I_0(\boldsymbol{x}) + \beta, \tag{8.8}$$

where β is the *bias* and α is the *gain* (Lucas and Kanade 1981; Gennert 1988; Fuh and Maragos 1991; Baker, Gross, and Matthews 2003; Evangelidis and Psarakis 2008). The least squares formulation then becomes

$$E_{\mathrm{BG}}(\boldsymbol{u}) = \sum_i [I_1(\boldsymbol{x}_i + \boldsymbol{u}) - (1 + \alpha) I_0(\boldsymbol{x}_i) - \beta]^2 = \sum_i [\alpha I_0(\boldsymbol{x}_i) + \beta - e_i]^2. \tag{8.9}$$

Rather than taking a simple squared difference between corresponding patches, it becomes necessary to perform a *linear regression* (Appendix A.2), which is somewhat more costly. Note that for color images, it may be necessary to estimate a different bias and gain for each color channel to compensate for the automatic *color correction* performed by some digital cameras (Section 2.3.2). Bias and gain compensation is also used in video codecs, where it is known as *weighted prediction* (Richardson 2003).

A more general (spatially varying, non-parametric) model of intensity variation, which is computed as part of the registration process, is used in (Negahdaripour 1998; Jia and Tang

2003; Seitz and Baker 2009). This can be useful for dealing with local variations such as the *vignetting* caused by wide-angle lenses, wide apertures, or lens housings. It is also possible to pre-process the images before comparing their values, e.g., using band-pass filtered images (Anandan 1989; Bergen, Anandan, Hanna *et al.* 1992), gradients (Scharstein 1994; Papenberg, Bruhn, Brox *et al.* 2006), or using other local transformations such as histograms or rank transforms (Cox, Roy, and Hingorani 1995; Zabih and Woodfill 1994), or to maximize *mutual information* (Viola and Wells III 1997; Kim, Kolmogorov, and Zabih 2003). Hirschmüller and Scharstein (2009) compare a number of these approaches and report on their relative performance in scenes with exposure differences.

Correlation. An alternative to taking intensity differences is to perform *correlation*, i.e., to maximize the *product* (or *cross-correlation*) of the two aligned images,

$$E_{\mathrm{CC}}(\boldsymbol{u}) = \sum_i I_0(\boldsymbol{x}_i) I_1(\boldsymbol{x}_i + \boldsymbol{u}). \tag{8.10}$$

At first glance, this may appear to make bias and gain modeling unnecessary, since the images will prefer to line up regardless of their relative scales and offsets. However, this is actually not true. If a very bright patch exists in $I_1(\boldsymbol{x})$, the maximum product may actually lie in that area.

For this reason, *normalized cross-correlation* is more commonly used,

$$E_{\mathrm{NCC}}(\boldsymbol{u}) = \frac{\sum_i [I_0(\boldsymbol{x}_i) - \overline{I_0}]\,[I_1(\boldsymbol{x}_i + \boldsymbol{u}) - \overline{I_1}]}{\sqrt{\sum_i [I_0(\boldsymbol{x}_i) - \overline{I_0}]^2}\sqrt{\sum_i [I_1(\boldsymbol{x}_i + \boldsymbol{u}) - \overline{I_1}]^2}}, \tag{8.11}$$

where

$$\overline{I_0} = \frac{1}{N}\sum_i I_0(\boldsymbol{x}_i) \quad \text{and} \tag{8.12}$$

$$\overline{I_1} = \frac{1}{N}\sum_i I_1(\boldsymbol{x}_i + \boldsymbol{u}) \tag{8.13}$$

are the *mean images* of the corresponding patches and N is the number of pixels in the patch. The normalized cross-correlation score is always guaranteed to be in the range $[-1, 1]$, which makes it easier to handle in some higher-level applications, such as deciding which patches truly match. Normalized correlation works well when matching images taken with different exposures, e.g., when creating high dynamic range images (Section 10.2). Note, however, that the NCC score is undefined if either of the two patches has zero variance (and, in fact, its performance degrades for noisy low-contrast regions).

A variant on NCC, which is related to the bias–gain regression implicit in the matching score (8.9), is the *normalized SSD* score

$$E_{\mathrm{NSSD}}(\boldsymbol{u}) = \frac{1}{2}\frac{\sum_i \big[[I_0(\boldsymbol{x}_i) - \overline{I_0}] - [I_1(\boldsymbol{x}_i + \boldsymbol{u}) - \overline{I_1}]\big]^2}{\sqrt{\sum_i [I_0(\boldsymbol{x}_i) - \overline{I_0}]^2 + [I_1(\boldsymbol{x}_i + \boldsymbol{u}) - \overline{I_1}]^2}} \tag{8.14}$$

recently proposed by Criminisi, Shotton, Blake *et al.* (2007). In their experiments, they find that it produces comparable results to NCC, but is more efficient when applied to a large number of overlapping patches using a moving average technique (Section 3.2.2).

8.1.1 Hierarchical motion estimation

Now that we have a well-defined alignment cost function to optimize, how can we find its minimum? The simplest solution is to do a *full search* over some range of shifts, using either integer or sub-pixel steps. This is often the approach used for *block matching* in *motion compensated video compression*, where a range of possible motions (say, ± 16 pixels) is explored.[4]

To accelerate this search process, *hierarchical motion estimation* is often used: an image pyramid (Section 3.5) is constructed and a search over a smaller number of discrete pixels (corresponding to the same range of motion) is first performed at coarser levels (Quam 1984; Anandan 1989; Bergen, Anandan, Hanna *et al.* 1992). The motion estimate from one level of the pyramid is then used to initialize a smaller *local* search at the next finer level. Alternatively, several seeds (good solutions) from the coarse level can be used to initialize the fine-level search. While this is not guaranteed to produce the same result as a full search, it usually works almost as well and is much faster.

More formally, let

$$I_k^{(l)}(\boldsymbol{x}_j) \leftarrow \tilde{I}_k^{(l-1)}(2\boldsymbol{x}_j) \tag{8.15}$$

be the *decimated* image at level l obtained by subsampling (*downsampling*) a smoothed version of the image at level $l-1$. See Section 3.5 for how to perform the required downsampling (pyramid construction) without introducing too much aliasing.

At the coarsest level, we search for the best displacement $\boldsymbol{u}^{(l)}$ that minimizes the difference between images $I_0^{(l)}$ and $I_1^{(l)}$. This is usually done using a full search over some range of displacements $\boldsymbol{u}^{(l)} \in 2^{-l}[-S, S]^2$, where S is the desired *search range* at the finest (original) resolution level, optionally followed by the incremental refinement step described in Section 8.1.3.

Once a suitable motion vector has been estimated, it is used to *predict* a likely displacement

$$\hat{\boldsymbol{u}}^{(l-1)} \leftarrow 2\boldsymbol{u}^{(l)} \tag{8.16}$$

for the next finer level.[5] The search over displacements is then repeated at the finer level over a much narrower range of displacements, say $\hat{\boldsymbol{u}}^{(l-1)} \pm 1$, again optionally combined with an incremental refinement step (Anandan 1989). Alternatively, one of the images can be *warped* (resampled) by the current motion estimate, in which case only small incremental motions need to be computed at the finer level. A nice description of the whole process, extended to parametric motion estimation (Section 8.2), is provided by Bergen, Anandan, Hanna *et al.* (1992).

8.1.2 Fourier-based alignment

When the search range corresponds to a significant fraction of the larger image (as is the case in image stitching, see Chapter 9), the hierarchical approach may not work that well, since

[4] In stereo matching (Section 11.1.2), an explicit search over all possible disparities (i.e., a *plane sweep*) is almost always performed, since the number of search hypotheses is much smaller due to the 1D nature of the potential displacements.

[5] This doubling of displacements is only necessary if displacements are defined in integer *pixel* coordinates, which is the usual case in the literature (Bergen, Anandan, Hanna *et al.* 1992). If *normalized device coordinates* (Section 2.1.5) are used instead, the displacements (and search ranges) need not change from level to level, although the step sizes will need to be adjusted, to keep search steps of roughly one pixel.

it is often not possible to coarsen the representation too much before significant features are blurred away. In this case, a Fourier-based approach may be preferable.

Fourier-based alignment relies on the fact that the Fourier transform of a shifted signal has the same magnitude as the original signal but a linearly varying phase (Section 3.4), i.e.,

$$\mathcal{F}\{I_1(\boldsymbol{x}+\boldsymbol{u})\} = \mathcal{F}\{I_1(\boldsymbol{x})\}\, e^{-ju\cdot\boldsymbol{\omega}} = \mathcal{I}_1(\boldsymbol{\omega})e^{-ju\cdot\boldsymbol{\omega}}, \tag{8.17}$$

where $\boldsymbol{\omega}$ is the vector-valued angular frequency of the Fourier transform and we use calligraphic notation $\mathcal{I}_1(\boldsymbol{\omega}) = \mathcal{F}\{I_1(\boldsymbol{x})\}$ to denote the Fourier transform of a signal (Section 3.4).

Another useful property of Fourier transforms is that convolution in the spatial domain corresponds to multiplication in the Fourier domain (Section 3.4).[6] Thus, the Fourier transform of the cross-correlation function E_{CC} can be written as

$$\mathcal{F}\{E_{\mathrm{CC}}(\boldsymbol{u})\} = \mathcal{F}\left\{\sum_i I_0(\boldsymbol{x}_i)I_1(\boldsymbol{x}_i+\boldsymbol{u})\right\} = \mathcal{F}\{I_0(\boldsymbol{u})\bar{*}I_1(\boldsymbol{u})\} = \mathcal{I}_0(\boldsymbol{\omega})\mathcal{I}_1^*(\boldsymbol{\omega}), \tag{8.18}$$

where

$$f(\boldsymbol{u})\bar{*}g(\boldsymbol{u}) = \sum_i f(\boldsymbol{x}_i)g(\boldsymbol{x}_i+\boldsymbol{u}) \tag{8.19}$$

is the *correlation* function, i.e., the convolution of one signal with the reverse of the other, and $\mathcal{I}_1^*(\boldsymbol{\omega})$ is the *complex conjugate* of $\mathcal{I}_1(\boldsymbol{\omega})$. This is because convolution is defined as the summation of one signal with the reverse of the other (Section 3.4).

Thus, to efficiently evaluate E_{CC} over the range of all possible values of $\boldsymbol{u}$, we take the Fourier transforms of both images $I_0(\boldsymbol{x})$ and $I_1(\boldsymbol{x})$, multiply both transforms together (after conjugating the second one), and take the inverse transform of the result. The Fast Fourier Transform algorithm can compute the transform of an $N \times M$ image in $\mathrm{O}(NM\log NM)$ operations (Bracewell 1986). This can be significantly faster than the $\mathrm{O}(N^2M^2)$ operations required to do a full search when the full range of image overlaps is considered.

While Fourier-based convolution is often used to accelerate the computation of image correlations, it can also be used to accelerate the sum of squared differences function (and its variants). Consider the SSD formula given in (8.1). Its Fourier transform can be written as

$$\begin{aligned}\mathcal{F}\{E_{\mathrm{SSD}}(\boldsymbol{u})\} &= \mathcal{F}\left\{\sum_i [I_1(\boldsymbol{x}_i+\boldsymbol{u}) - I_0(\boldsymbol{x}_i)]^2\right\} \\ &= \delta(\boldsymbol{\omega})\sum_i [I_0^2(\boldsymbol{x}_i) + I_1^2(\boldsymbol{x}_i)] - 2\mathcal{I}_0(\boldsymbol{\omega})\mathcal{I}_1^*(\boldsymbol{\omega}).\end{aligned} \tag{8.20}$$

Thus, the SSD function can be computed by taking twice the correlation function and subtracting it from the sum of the energies in the two images.

Windowed correlation. Unfortunately, the Fourier convolution theorem only applies when the summation over $\boldsymbol{x}_i$ is performed over *all* the pixels in both images, using a circular shift of the image when accessing pixels outside the original boundaries. While this is

[6] In fact, the Fourier shift property (8.17) derives from the convolution theorem by observing that shifting is equivalent to convolution with a displaced delta function $\delta(\boldsymbol{x}-\boldsymbol{u})$.

acceptable for small shifts and comparably sized images, it makes no sense when the images overlap by a small amount or one image is a small subset of the other.

In that case, the cross-correlation function should be replaced with a *windowed* (weighted) cross-correlation function,

$$E_{\mathrm{WCC}}(\boldsymbol{u}) = \sum_i w_0(\boldsymbol{x}_i) I_0(\boldsymbol{x}_i)\, w_1(\boldsymbol{x}_i + \boldsymbol{u}) I_1(\boldsymbol{x}_i + \boldsymbol{u}), \tag{8.21}$$

$$= [w_0(\boldsymbol{x}) I_0(\boldsymbol{x})] \bar{*} [w_1(\boldsymbol{x}) I_1(\boldsymbol{x})] \tag{8.22}$$

where the weighting functions w_0 and w_1 are zero outside the valid ranges of the images and both images are padded so that circular shifts return 0 values outside the original image boundaries.

An even more interesting case is the computation of the *weighted* SSD function introduced in Equation (8.5),

$$E_{\mathrm{WSSD}}(\boldsymbol{u}) = \sum_i w_0(\boldsymbol{x}_i) w_1(\boldsymbol{x}_i + \boldsymbol{u}) [I_1(\boldsymbol{x}_i + \boldsymbol{u}) - I_0(\boldsymbol{x}_i)]^2. \tag{8.23}$$

Expanding this as a sum of correlations and deriving the appropriate set of Fourier transforms is left for Exercise 8.1.

The same kind of derivation can also be applied to the bias–gain corrected sum of squared difference function E_{BG} (8.9). Again, Fourier transforms can be used to efficiently compute all the correlations needed to perform the linear regression in the bias and gain parameters in order to estimate the exposure-compensated difference for each potential shift (Exercise 8.1).

Phase correlation. A variant of regular correlation (8.18) that is sometimes used for motion estimation is *phase correlation* (Kuglin and Hines 1975; Brown 1992). Here, the spectrum of the two signals being matched is *whitened* by dividing each per-frequency product in (8.18) by the magnitudes of the Fourier transforms,

$$\mathcal{F}\{E_{\mathrm{PC}}(\boldsymbol{u})\} = \frac{\mathcal{I}_0(\boldsymbol{\omega}) \mathcal{I}_1^*(\boldsymbol{\omega})}{\|\mathcal{I}_0(\boldsymbol{\omega})\| \|\mathcal{I}_1(\boldsymbol{\omega})\|} \tag{8.24}$$

before taking the final inverse Fourier transform. In the case of noiseless signals with perfect (cyclic) shift, we have $I_1(\boldsymbol{x} + \boldsymbol{u}) = I_0(\boldsymbol{x})$ and hence, from Equation (8.17), we obtain

$$\mathcal{F}\{I_1(\boldsymbol{x} + \boldsymbol{u})\} = \mathcal{I}_1(\boldsymbol{\omega}) e^{-2\pi j u \cdot \boldsymbol{\omega}} = \mathcal{I}_0(\boldsymbol{\omega}) \text{ and}$$

$$\mathcal{F}\{E_{\mathrm{PC}}(\boldsymbol{u})\} = e^{-2\pi j u \cdot \boldsymbol{\omega}}. \tag{8.25}$$

The output of phase correlation (under ideal conditions) is therefore a single spike (impulse) located at the correct value of $\boldsymbol{u}$, which (in principle) makes it easier to find the correct estimate.

Phase correlation has a reputation in some quarters of outperforming regular correlation, but this behavior depends on the characteristics of the signals and noise. If the original images are contaminated by noise in a narrow frequency band (e.g., low-frequency noise or peaked frequency "hum"), the whitening process effectively de-emphasizes the noise in these regions. However, if the original signals have very low signal-to-noise ratio at some frequencies (say, two blurry or low-textured images with lots of high-frequency noise), the whitening process can actually decrease performance (see Exercise 8.1).

Recently, gradient cross-correlation has emerged as a promising alternative to phase correlation (Argyriou and Vlachos 2003), although further systematic studies are probably warranted. Phase correlation has also been studied by Fleet and Jepson (1990) as a method for estimating general optical flow and stereo disparity.

Rotations and scale. While Fourier-based alignment is mostly used to estimate translational shifts between images, it can, under certain limited conditions, also be used to estimate in-plane rotations and scales. Consider two images that are related *purely* by rotation, i.e.,

$$I_1(\hat{\boldsymbol{R}}\boldsymbol{x}) = I_0(\boldsymbol{x}). \tag{8.26}$$

If we re-sample the images into *polar coordinates*,

$$\tilde{I}_0(r,\theta) = I_0(r\cos\theta, r\sin\theta) \;\text{ and }\; \tilde{I}_1(r,\theta) = I_1(r\cos\theta, r\sin\theta), \tag{8.27}$$

we obtain

$$\tilde{I}_1(r,\theta+\hat{\theta}) = \tilde{I}_0(r,\theta). \tag{8.28}$$

The desired rotation can then be estimated using a Fast Fourier Transform (FFT) shift-based technique.

If the two images are also related by a scale,

$$I_1(e^{\hat{s}}\hat{\boldsymbol{R}}\boldsymbol{x}) = I_0(\boldsymbol{x}), \tag{8.29}$$

we can re-sample into *log-polar coordinates*,

$$\tilde{I}_0(s,\theta) = I_0(e^s\cos\theta, e^s\sin\theta) \;\text{ and }\; \tilde{I}_1(s,\theta) = I_1(e^s\cos\theta, e^s\sin\theta), \tag{8.30}$$

to obtain

$$\tilde{I}_1(s+\hat{s},\theta+\hat{\theta}) = I_0(s,\theta). \tag{8.31}$$

In this case, care must be taken to choose a suitable range of s values that reasonably samples the original image.

For images that are also translated by a small amount,

$$I_1(e^{\hat{s}}\hat{\boldsymbol{R}}\boldsymbol{x}+\boldsymbol{t}) = I_0(\boldsymbol{x}), \tag{8.32}$$

De Castro and Morandi (1987) propose an ingenious solution that uses several steps to estimate the unknown parameters. First, both images are converted to the Fourier domain and only the magnitudes of the transformed images are retained. In principle, the Fourier magnitude images are insensitive to translations in the image plane (although the usual caveats about border effects apply). Next, the two magnitude images are aligned in rotation and scale using the polar or log-polar representations. Once rotation and scale are estimated, one of the images can be de-rotated and scaled and a regular translational algorithm can be applied to estimate the translational shift.

Unfortunately, this trick only applies when the images have large overlap (small translational motion). For more general motion of patches or images, the parametric motion estimator described in Section 8.2 or the feature-based approaches described in Section 6.1 need to be used.

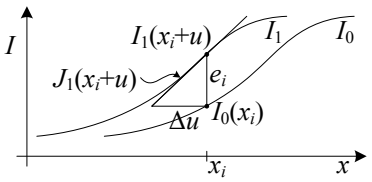


Figure 8.2 Taylor series approximation of a function and the incremental computation of the optical flow correction amount. $\boldsymbol{J}_1(\boldsymbol{x}_i + \boldsymbol{u})$ is the image gradient at $(\boldsymbol{x}_i + \boldsymbol{u})$ and e_i is the current intensity difference.

8.1.3 Incremental refinement

The techniques described up till now can estimate alignment to the nearest pixel (or potentially fractional pixel if smaller search steps are used). In general, image stabilization and stitching applications require much higher accuracies to obtain acceptable results.

To obtain better *sub-pixel* estimates, we can use one of several techniques described by Tian and Huhns (1986). One possibility is to evaluate several discrete (integer or fractional) values of (u, v) around the best value found so far and to *interpolate* the matching score to find an analytic minimum.

A more commonly used approach, first proposed by Lucas and Kanade (1981), is to perform *gradient descent* on the SSD energy function (8.1), using a Taylor series expansion of the image function (Figure 8.2),

$$E_{\mathrm{LK-SSD}}(\boldsymbol{u} + \Delta\boldsymbol{u}) = \sum_i [I_1(\boldsymbol{x}_i + \boldsymbol{u} + \Delta\boldsymbol{u}) - I_0(\boldsymbol{x}_i)]^2 \tag{8.33}$$

$$\approx \sum_i [I_1(\boldsymbol{x}_i + \boldsymbol{u}) + \boldsymbol{J}_1(\boldsymbol{x}_i + \boldsymbol{u})\Delta\boldsymbol{u} - I_0(\boldsymbol{x}_i)]^2 \tag{8.34}$$

$$= \sum_i [\boldsymbol{J}_1(\boldsymbol{x}_i + \boldsymbol{u})\Delta\boldsymbol{u} + e_i]^2, \tag{8.35}$$

where

$$\boldsymbol{J}_1(\boldsymbol{x}_i + \boldsymbol{u}) = \nabla I_1(\boldsymbol{x}_i + \boldsymbol{u}) = (\frac{\partial I_1}{\partial x}, \frac{\partial I_1}{\partial y})(\boldsymbol{x}_i + \boldsymbol{u}) \tag{8.36}$$

is the *image gradient* or *Jacobian* at $(\boldsymbol{x}_i + \boldsymbol{u})$ and

$$e_i = I_1(\boldsymbol{x}_i + \boldsymbol{u}) - I_0(\boldsymbol{x}_i), \tag{8.37}$$

first introduced in (8.1), is the current intensity error.[7] The gradient at a particular sub-pixel location $(\boldsymbol{x}_i + \boldsymbol{u})$ can be computed using a variety of techniques, the simplest of which is to simply take the horizontal and vertical differences between pixels $\boldsymbol{x}$ and $\boldsymbol{x} + (1, 0)$ or $\boldsymbol{x} + (0, 1)$. More sophisticated derivatives can sometimes lead to noticeable performance improvements.

The linearized form of the incremental update to the SSD error (8.35) is often called the *optical flow constraint* or *brightness constancy constraint* equation

$$I_x u + I_y v + I_t = 0, \tag{8.38}$$

[7] We follow the convention, commonly used in robotics and by Baker and Matthews (2004), that derivatives with respect to (column) vectors result in row vectors, so that fewer transposes are needed in the formulas.

where the subscripts in I_x and I_y denote spatial derivatives, and I_t is called the *temporal derivative*, which makes sense if we are computing instantaneous velocity in a video sequence. When squared and summed or integrated over a region, it can be used to compute optic flow (Horn and Schunck 1981).

The above least squares problem (8.35) can be minimized by solving the associated *normal equations* (Appendix A.2),

$$\boldsymbol{A}\Delta\boldsymbol{u} = \boldsymbol{b} \tag{8.39}$$

where

$$\boldsymbol{A} = \sum_i \boldsymbol{J}_1^T(\boldsymbol{x}_i + \boldsymbol{u})\boldsymbol{J}_1(\boldsymbol{x}_i + \boldsymbol{u}) \tag{8.40}$$

and

$$\boldsymbol{b} = -\sum_i e_i \boldsymbol{J}_1^T(\boldsymbol{x}_i + \boldsymbol{u}) \tag{8.41}$$

are called the (Gauss–Newton approximation of the) *Hessian* and *gradient-weighted residual vector*, respectively.[8] These matrices are also often written as

$$\boldsymbol{A} = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} \text{ and } \boldsymbol{b} = -\begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}. \tag{8.42}$$

The gradients required for $\boldsymbol{J}_1(\boldsymbol{x}_i + \boldsymbol{u})$ can be evaluated at the same time as the image warps required to estimate $I_1(\boldsymbol{x}_i + \boldsymbol{u})$ (Section 3.6.1 (3.89)) and, in fact, are often computed as a side-product of image interpolation. If efficiency is a concern, these gradients can be replaced by the gradients in the *template* image,

$$\boldsymbol{J}_1(\boldsymbol{x}_i + \boldsymbol{u}) \approx \boldsymbol{J}_0(\boldsymbol{x}_i), \tag{8.43}$$

since near the correct alignment, the template and displaced target images should look similar. This has the advantage of allowing the pre-computation of the Hessian and Jacobian images, which can result in significant computational savings (Hager and Belhumeur 1998; Baker and Matthews 2004). A further reduction in computation can be obtained by writing the warped image $I_1(\boldsymbol{x}_i + \boldsymbol{u})$ used to compute e_i in (8.37) as a convolution of a sub-pixel interpolation filter with the discrete samples in I_1 (Peleg and Rav-Acha 2006). Precomputing the inner product between the gradient field and shifted version of I_1 allows the iterative re-computation of e_i to be performed in constant time (independent of the number of pixels).

The effectiveness of the above incremental update rule relies on the quality of the Taylor series approximation. When far away from the true displacement (say, 1–2 pixels), several iterations may be needed. It is possible, however, to estimate a value for $\boldsymbol{J}_1$ using a least squares fit to a series of larger displacements in order to increase the range of convergence (Jurie and Dhome 2002) or to "learn" a special-purpose recognizer for a given patch (Avidan 2001; Williams, Blake, and Cipolla 2003; Lepetit, Pilet, and Fua 2006; Hinterstoisser, Benhimane, Navab *et al.* 2008; Özuysal, Calonder, Lepetit *et al.* 2010) as discussed in Section 4.1.4.

A commonly used stopping criterion for incremental updating is to monitor the magnitude of the displacement correction $\|\boldsymbol{u}\|$ and to stop when it drops below a certain threshold (say,

[8] The true Hessian is the full second derivative of the error function E, which may not be positive definite—see Section 6.1.3 and Appendix A.3.

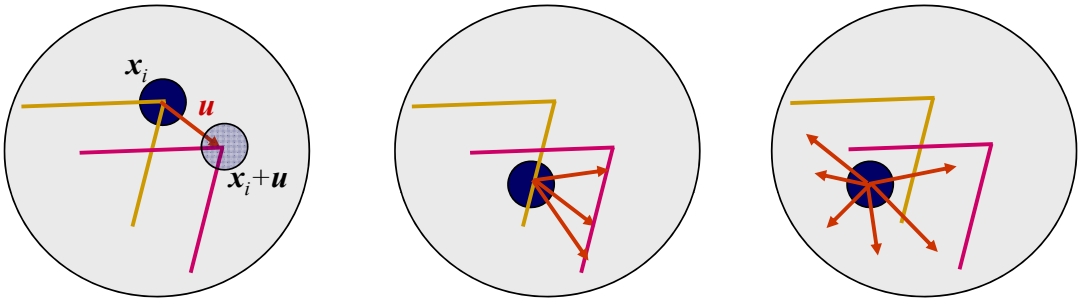


Figure 8.3 Aperture problems for different image regions, denoted by the orange and red L-shaped structures, overlaid in the same image to make it easier to diagram the flow. (a) A window $w(\boldsymbol{x}_i)$ centered at $\boldsymbol{x}_i$ (black circle) can uniquely be matched to its corresponding structure at $\boldsymbol{x}_i + \boldsymbol{u}$ in the second (red) image. (b) A window centered on the edge exhibits the classic aperture problem, since it can be matched to a 1D family of possible locations. (c) In a completely textureless region, the matches become totally unconstrained.

$^1/_{10}$ of a pixel). For larger motions, it is usual to combine the incremental update rule with a hierarchical coarse-to-fine search strategy, as described in Section 8.1.1.

Conditioning and aperture problems. Sometimes, the inversion of the linear system (8.39) can be poorly conditioned because of lack of two-dimensional texture in the patch being aligned. A commonly occurring example of this is the *aperture problem*, first identified in some of the early papers on optical flow (Horn and Schunck 1981) and then studied more extensively by Anandan (1989). Consider an image patch that consists of a slanted edge moving to the right (Figure 8.3). Only the *normal* component of the velocity (displacement) can be reliably recovered in this case. This manifests itself in (8.39) as a *rank-deficient* matrix $\boldsymbol{A}$, i.e., one whose smaller eigenvalue is very close to zero.[9]

When Equation (8.39) is solved, the component of the displacement along the edge is very poorly conditioned and can result in wild guesses under small noise perturbations. One way to mitigate this problem is to add a *prior* (soft constraint) on the expected range of motions (Simoncelli, Adelson, and Heeger 1991; Baker, Gross, and Matthews 2004; Govindu 2006). This can be accomplished by adding a small value to the diagonal of $\boldsymbol{A}$, which essentially biases the solution towards smaller $\Delta\boldsymbol{u}$ values that still (mostly) minimize the squared error.

However, the pure Gaussian model assumed when using a simple (fixed) quadratic prior, as in (Simoncelli, Adelson, and Heeger 1991), does not always hold in practice, e.g., because of aliasing along strong edges (Triggs 2004). For this reason, it may be prudent to add some small fraction (say, 5%) of the larger eigenvalue to the smaller one before doing the matrix inversion.

Uncertainty modeling. The reliability of a particular patch-based motion estimate can be captured more formally with an *uncertainty model*. The simplest such model is a *covariance matrix*, which captures the expected variance in the motion estimate in all possible directions.

[9] The matrix $\boldsymbol{A}$ is by construction always guaranteed to be symmetric positive semi-definite, i.e., it has real non-negative eigenvalues.

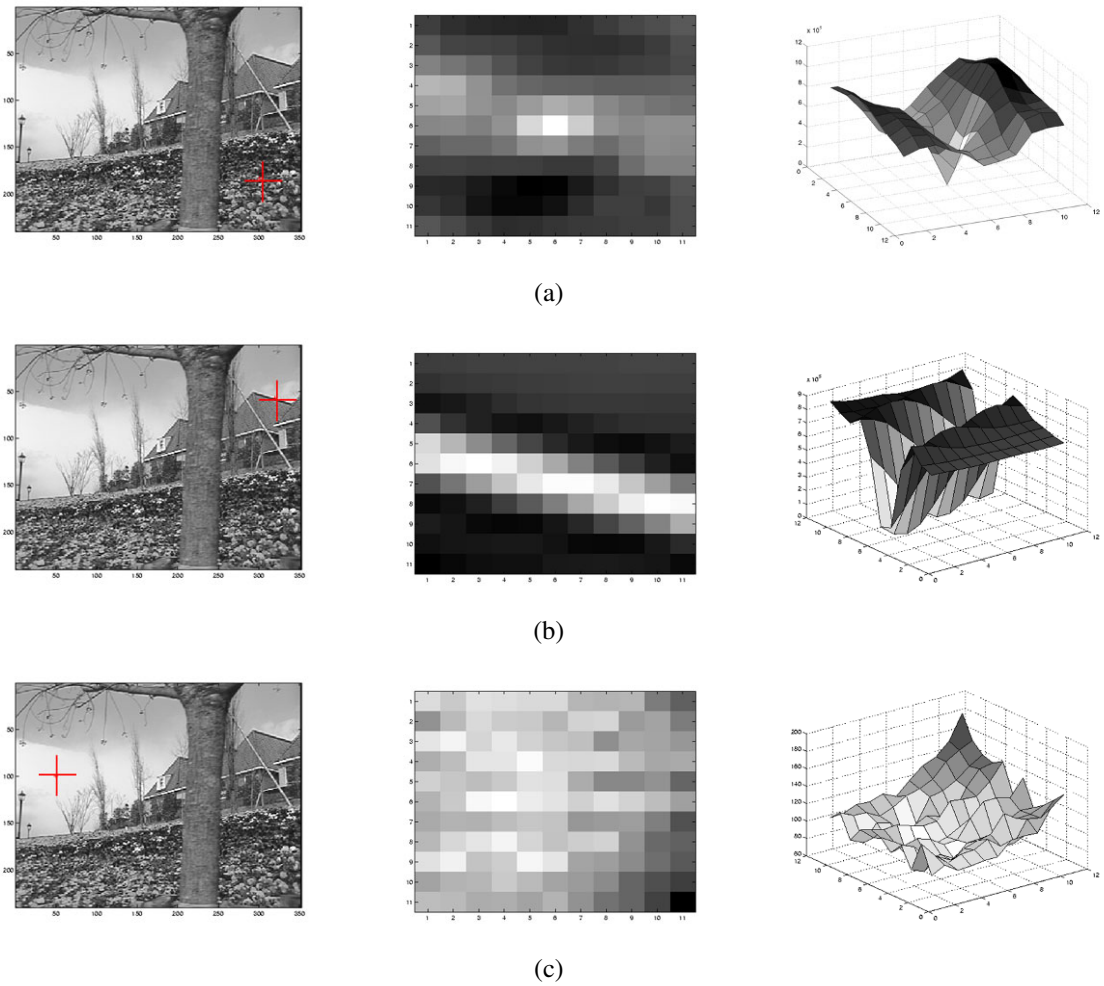


Figure 8.4 SSD surfaces corresponding to three locations (red crosses) in an image: (a) highly textured area, strong minimum, low uncertainty; (b) strong edge, aperture problem, high uncertainty in one direction; (c) weak texture, no clear minimum, large uncertainty.

As discussed in Section 6.1.4 and Appendix B.6, under small amounts of additive Gaussian noise, it can be shown that the covariance matrix $\Sigma_{\boldsymbol{u}}$ is proportional to the inverse of the Hessian $\boldsymbol{A}$,

$$\Sigma_{\boldsymbol{u}} = \sigma_n^2 \boldsymbol{A}^{-1}, \tag{8.44}$$

where σ_n^2 is the variance of the additive Gaussian noise (Anandan 1989; Matthies, Kanade, and Szeliski 1989; Szeliski 1989).

For larger amounts of noise, the linearization performed by the Lucas–Kanade algorithm in (8.35) is only approximate, so the above quantity becomes a *Cramer–Rao lower bound* on the true covariance. Thus, the minimum and maximum eigenvalues of the Hessian $\boldsymbol{A}$ can now be interpreted as the (scaled) inverse variances in the least-certain and most-certain directions of motion. (A more detailed analysis using a more realistic model of image noise is given by Steele and Jaynes (2005).) Figure 8.4 shows the local SSD surfaces for three different pixel locations in an image. As you can see, the surface has a clear minimum in the highly textured region and suffers from the aperture problem near the strong edge.

Bias and gain, weighting, and robust error metrics. The Lucas–Kanade update rule can also be applied to the bias–gain equation (8.9) to obtain

$$E_{\mathrm{LK-BG}}(\boldsymbol{u} + \Delta\boldsymbol{u}) = \sum_i [\boldsymbol{J}_1(\boldsymbol{x}_i + \boldsymbol{u})\Delta\boldsymbol{u} + e_i - \alpha I_0(\boldsymbol{x}_i) - \beta]^2 \tag{8.45}$$

(Lucas and Kanade 1981; Gennert 1988; Fuh and Maragos 1991; Baker, Gross, and Matthews 2003). The resulting 4×4 system of equations can be solved to simultaneously estimate the translational displacement update $\Delta\boldsymbol{u}$ and the bias and gain parameters β and α.

A similar formulation can be derived for images (templates) that have a *linear appearance variation*,

$$I_1(\boldsymbol{x} + \boldsymbol{u}) \approx I_0(\boldsymbol{x}) + \sum_j \lambda_j B_j(\boldsymbol{x}), \tag{8.46}$$

where the $B_j(\boldsymbol{x})$ are the *basis images* and the λ_j are the unknown coefficients (Hager and Belhumeur 1998; Baker, Gross, Ishikawa *et al.* 2003; Baker, Gross, and Matthews 2003). Potential linear appearance variations include illumination changes (Hager and Belhumeur 1998) and small non-rigid deformations (Black and Jepson 1998).

A weighted (windowed) version of the Lucas–Kanade algorithm is also possible:

$$E_{\mathrm{LK-WSSD}}(\boldsymbol{u} + \Delta\boldsymbol{u}) = \sum_i w_0(\boldsymbol{x}_i) w_1(\boldsymbol{x}_i + \boldsymbol{u})[\boldsymbol{J}_1(\boldsymbol{x}_i + \boldsymbol{u})\Delta\boldsymbol{u} + e_i]^2. \tag{8.47}$$

Note that here, in deriving the Lucas–Kanade update from the original weighted SSD function (8.5), we have neglected taking the derivative of the $w_1(\boldsymbol{x}_i + \boldsymbol{u})$ weighting function with respect to $\boldsymbol{u}$, which is usually acceptable in practice, especially if the weighting function is a binary mask with relatively few transitions.

Baker, Gross, Ishikawa *et al.* (2003) only use the $w_0(\boldsymbol{x})$ term, which is reasonable if the two images have the same extent and no (independent) cutouts in the overlap region. They also discuss the idea of making the weighting proportional to $\nabla I(\boldsymbol{x})$, which helps for very noisy images, where the gradient itself is noisy. Similar observations, formulated in terms of *total least squares* (Van Huffel and Vandewalle 1991; Van Huffel and Lemmerling 2002),

have been made by other researchers studying optical flow (Weber and Malik 1995; Bab-Hadiashar and Suter 1998b; Mühlich and Mester 1998). Lastly, Baker, Gross, Ishikawa *et al.* (2003) show how evaluating Equation (8.47) at just the *most reliable* (highest gradient) pixels does not significantly reduce performance for large enough images, even if only 5–10% of the pixels are used. (This idea was originally proposed by Dellaert and Collins (1999), who used a more sophisticated selection criterion.)

The Lucas–Kanade incremental refinement step can also be applied to the robust error metric introduced in Section 8.1,

$$E_{\mathrm{LK-SRD}}(\boldsymbol{u}+\Delta\boldsymbol{u}) = \sum_i \rho(\boldsymbol{J}_1(\boldsymbol{x}_i+\boldsymbol{u})\Delta\boldsymbol{u} + e_i), \tag{8.48}$$

which can be solved using the *iteratively reweighted least squares* technique described in Section 6.1.4.

8.2 Parametric motion

Many image alignment tasks, for example image stitching with handheld cameras, require the use of more sophisticated motion models, as described in Section 2.1.2. Since these models, e.g., affine deformations, typically have more parameters than pure translation, a full search over the possible range of values is impractical. Instead, the incremental Lucas–Kanade algorithm can be generalized to parametric motion models and used in conjunction with a hierarchical search algorithm (Lucas and Kanade 1981; Rehg and Witkin 1991; Fuh and Maragos 1991; Bergen, Anandan, Hanna *et al.* 1992; Shashua and Toelg 1997; Shashua and Wexler 2001; Baker and Matthews 2004).

For parametric motion, instead of using a single constant translation vector $\boldsymbol{u}$, we use a spatially varying *motion field* or *correspondence map*, $\boldsymbol{x}'(\boldsymbol{x};\boldsymbol{p})$, parameterized by a low-dimensional vector $\boldsymbol{p}$, where $\boldsymbol{x}'$ can be any of the motion models presented in Section 2.1.2. The parametric incremental motion update rule now becomes

$$\begin{aligned} E_{\mathrm{LK-PM}}(\boldsymbol{p}+\Delta\boldsymbol{p}) &= \sum_i [I_1(\boldsymbol{x}'(\boldsymbol{x}_i;\boldsymbol{p}+\Delta\boldsymbol{p})) - I_0(\boldsymbol{x}_i)]^2 && (8.49)\\ &\approx \sum_i [I_1(\boldsymbol{x}'_i) + \boldsymbol{J}_1(\boldsymbol{x}'_i)\Delta\boldsymbol{p} - I_0(\boldsymbol{x}_i)]^2 && (8.50)\\ &= \sum_i [\boldsymbol{J}_1(\boldsymbol{x}'_i)\Delta\boldsymbol{p} + e_i]^2, && (8.51) \end{aligned}$$

where the Jacobian is now

$$\boldsymbol{J}_1(\boldsymbol{x}'_i) = \frac{\partial I_1}{\partial \boldsymbol{p}} = \nabla I_1(\boldsymbol{x}'_i)\frac{\partial \boldsymbol{x}'}{\partial \boldsymbol{p}}(\boldsymbol{x}_i), \tag{8.52}$$

i.e., the product of the image gradient ∇I_1 with the Jacobian of the correspondence field, $\boldsymbol{J}_{x'} = \partial \boldsymbol{x}'/\partial \boldsymbol{p}$.

The motion Jacobians $\boldsymbol{J}_{\boldsymbol{x}'}$ for the 2D planar transformations introduced in Section 2.1.2 and Table 2.1 are given in Table 6.1. Note how we have re-parameterized the motion matrices so that they are always the identity at the origin $\boldsymbol{p} = 0$. This becomes useful later, when we talk about the compositional and inverse compositional algorithms. (It also makes it easier to impose priors on the motions.)

For parametric motion, the (Gauss–Newton) *Hessian* and *gradient-weighted residual vector* become

$$\boldsymbol{A} = \sum_i \boldsymbol{J}_{\boldsymbol{x}'}^T(\boldsymbol{x}_i)[\nabla I_1^T(\boldsymbol{x}_i')\nabla I_1(\boldsymbol{x}_i')]\boldsymbol{J}_{\boldsymbol{x}'}(\boldsymbol{x}_i) \tag{8.53}$$

and

$$\boldsymbol{b} = -\sum_i \boldsymbol{J}_{\boldsymbol{x}'}^T(\boldsymbol{x}_i)[e_i \nabla I_1^T(\boldsymbol{x}_i')]. \tag{8.54}$$

Note how the expressions inside the square brackets are the same ones evaluated for the simpler translational motion case (8.40–8.41).

Patch-based approximation. The computation of the Hessian and residual vectors for parametric motion can be significantly more expensive than for the translational case. For parametric motion with n parameters and N pixels, the accumulation of $\boldsymbol{A}$ and $\boldsymbol{b}$ takes $O(n^2N)$ operations (Baker and Matthews 2004). One way to reduce this by a significant amount is to divide the image up into smaller sub-blocks (patches) P_j and to only accumulate the simpler 2×2 quantities inside the square brackets at the pixel level (Shum and Szeliski 2000),

$$\boldsymbol{A}_j = \sum_{i \in P_j} \nabla I_1^T(\boldsymbol{x}_i')\nabla I_1(\boldsymbol{x}_i') \tag{8.55}$$

$$\boldsymbol{b}_j = \sum_{i \in P_j} e_i \nabla I_1^T(\boldsymbol{x}_i'). \tag{8.56}$$

The full Hessian and residual can then be approximated as

$$\boldsymbol{A} \approx \sum_j \boldsymbol{J}_{\boldsymbol{x}'}^T(\hat{\boldsymbol{x}}_j)[\sum_{i \in P_j} \nabla I_1^T(\boldsymbol{x}_i')\nabla I_1(\boldsymbol{x}_i')]\boldsymbol{J}_{\boldsymbol{x}'}(\hat{\boldsymbol{x}}_j) = \sum_j \boldsymbol{J}_{\boldsymbol{x}'}^T(\hat{\boldsymbol{x}}_j)\boldsymbol{A}_j\boldsymbol{J}_{\boldsymbol{x}'}(\hat{\boldsymbol{x}}_j) \tag{8.57}$$

and

$$\boldsymbol{b} \approx -\sum_j \boldsymbol{J}_{\boldsymbol{x}'}^T(\hat{\boldsymbol{x}}_j)[\sum_{i \in P_j} e_i \nabla I_1^T(\boldsymbol{x}_i')] = -\sum_j \boldsymbol{J}_{\boldsymbol{x}'}^T(\hat{\boldsymbol{x}}_j)\boldsymbol{b}_j, \tag{8.58}$$

where $\hat{\boldsymbol{x}}_j$ is the *center* of each patch P_j (Shum and Szeliski 2000). This is equivalent to replacing the true motion Jacobian with a piecewise-constant approximation. In practice, this works quite well. The relationship of this approximation to feature-based registration is discussed in Section 9.2.4.

Compositional approach. For a complex parametric motion such as a homography, the computation of the motion Jacobian becomes complicated and may involve a per-pixel division. Szeliski and Shum (1997) observed that this can be simplified by first warping the target image I_1 according to the current motion estimate $\boldsymbol{x}'(\boldsymbol{x};\boldsymbol{p})$,

$$\tilde{I}_1(\boldsymbol{x}) = I_1(\boldsymbol{x}'(\boldsymbol{x};\boldsymbol{p})), \tag{8.59}$$

and then comparing this *warped* image against the template $I_0(\boldsymbol{x})$,

$$E_{\mathrm{LK-SS}}(\Delta\boldsymbol{p}) = \sum_i [\tilde{I}_1(\tilde{\boldsymbol{x}}(\boldsymbol{x}_i;\Delta\boldsymbol{p})) - I_0(\boldsymbol{x}_i)]^2 \tag{8.60}$$

$$\approx \sum_i [\tilde{J}_1(\boldsymbol{x}_i)\Delta\boldsymbol{p} + e_i]^2 \tag{8.61}$$

$$= \sum_i [\nabla\tilde{I}_1(\boldsymbol{x}_i)\boldsymbol{J}_{\tilde{\boldsymbol{x}}}(\boldsymbol{x}_i)\Delta\boldsymbol{p} + e_i]^2. \tag{8.62}$$

Note that since the two images are assumed to be fairly similar, only an *incremental* parametric motion is required, i.e., the incremental motion can be evaluated around $\boldsymbol{p} = 0$, which can lead to considerable simplifications. For example, the Jacobian of the planar projective transform (6.19) now becomes

$$\boldsymbol{J}_{\tilde{\boldsymbol{x}}} = \frac{\partial \tilde{\boldsymbol{x}}}{\partial \boldsymbol{p}}\bigg|_{\boldsymbol{p}=0} = \begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -x^2 & -xy \\ 0 & 0 & 0 & x & y & 1 & -xy & -y^2 \end{bmatrix}. \tag{8.63}$$

Once the incremental motion $\tilde{\boldsymbol{x}}$ has been computed, it can be *prepended* to the previously estimated motion, which is easy to do for motions represented with transformation matrices, such as those given in Tables 2.1 and 6.1. Baker and Matthews (2004) call this the *forward compositional* algorithm, since the target image is being re-warped and the final motion estimates are being composed.

If the appearance of the warped and template images is similar enough, we can replace the gradient of $\tilde{I}_1(\boldsymbol{x})$ with the gradient of $I_0(\boldsymbol{x})$, as suggested previously (8.43). This has potentially a big advantage in that it allows the pre-computation (and inversion) of the Hessian matrix $\boldsymbol{A}$ given in Equation (8.53). The residual vector $\boldsymbol{b}$ (8.54) can also be partially precomputed, i.e., the *steepest descent images* $\nabla I_0(\boldsymbol{x})\boldsymbol{J}_{\tilde{\boldsymbol{x}}}(\boldsymbol{x})$ can precomputed and stored for later multiplication with the $e(\boldsymbol{x}) = \tilde{I}_1(\boldsymbol{x}) - I_0(\boldsymbol{x})$ error images (Baker and Matthews 2004). This idea was first suggested by Hager and Belhumeur (1998) in what Baker and Matthews (2004) call a *inverse additive* scheme.

Baker and Matthews (2004) introduce one more variant they call the *inverse compositional* algorithm. Rather than (conceptually) re-warping the warped target image $\tilde{I}_1(\boldsymbol{x})$, they instead warp the template image $I_0(\boldsymbol{x})$ and minimize

$$E_{\mathrm{LK-BM}}(\Delta\boldsymbol{p}) = \sum_i [\tilde{I}_1(\boldsymbol{x}_i) - I_0(\tilde{\boldsymbol{x}}(\boldsymbol{x}_i; \Delta\boldsymbol{p}))]^2 \tag{8.64}$$

$$\approx \sum_i [\nabla I_0(\boldsymbol{x}_i)\boldsymbol{J}_{\tilde{\boldsymbol{x}}}(\boldsymbol{x}_i)\Delta\boldsymbol{p} - e_i]^2. \tag{8.65}$$

This is identical to the forward warped algorithm (8.62) with the gradients $\nabla\tilde{I}_1(\boldsymbol{x})$ replaced by the gradients $\nabla I_0(\boldsymbol{x})$, except for the sign of e_i. The resulting update $\Delta\boldsymbol{p}$ is the *negative* of the one computed by the modified Equation (8.62) and hence the *inverse* of the incremental transformation must be prepended to the current transform. Because the inverse compositional algorithm has the potential of pre-computing the inverse Hessian and the steepest descent images, this makes it the preferred approach of those surveyed by Baker and Matthews (2004). Figure 8.5 (Baker, Gross, Ishikawa *et al.* 2003) beautifully shows all of the steps required to implement the inverse compositional algorithm.

Baker and Matthews (2004) also discuss the advantage of using Gauss–Newton iteration (i.e., the first-order expansion of the least squares, as above) compared to other approaches such as steepest descent and Levenberg–Marquardt. Subsequent parts of the series (Baker, Gross, Ishikawa *et al.* 2003; Baker, Gross, and Matthews 2003, 2004) discuss more advanced topics such as per-pixel weighting, pixel selection for efficiency, a more in-depth discussion of robust metrics and algorithms, linear appearance variations, and priors on parameters. They make for invaluable reading for anyone interested in implementing a highly tuned implementation of incremental image registration. Evangelidis and Psarakis (2008) provide some detailed experimental evaluations of these and other related approaches.

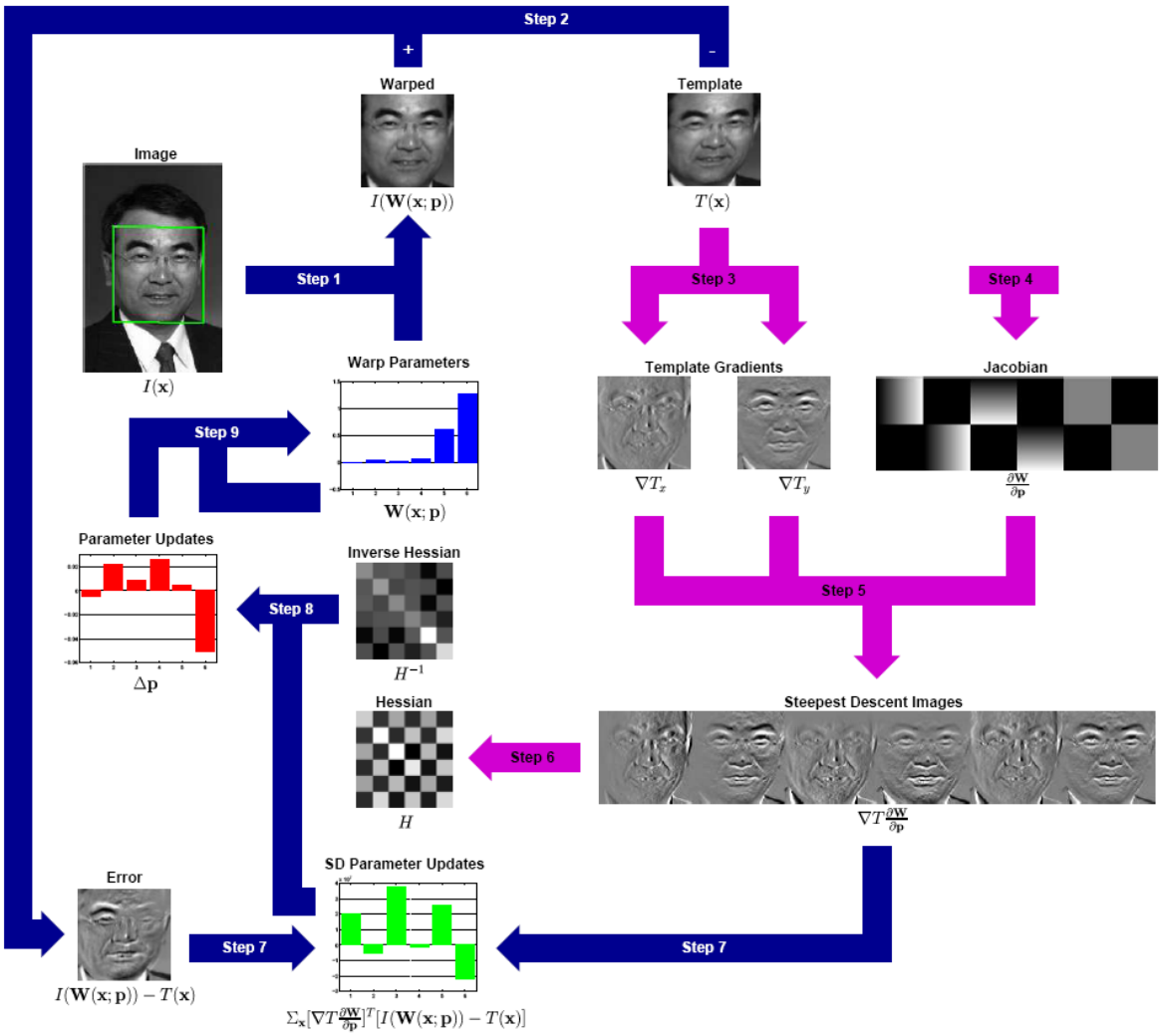


Figure 8.5 A schematic overview of the inverse compositional algorithm (copied, with permission, from (Baker, Gross, Ishikawa *et al.* 2003)). Steps 3–6 (light-colored arrows) are performed once as a pre-computation. The main algorithm simply consists of iterating: image warping (Step 1), image differencing (Step 2), image dot products (Step 7), multiplication with the inverse of the Hessian (Step 8), and the update to the warp (Step 9). All of these steps can be performed efficiently.

8.2.1 *Application*: Video stabilization

Video stabilization is one of the most widely used applications of parametric motion estimation (Hansen, Anandan, Dana *et al.* 1994; Irani, Rousso, and Peleg 1997; Morimoto and Chellappa 1997; Srinivasan, Chellappa, Veeraraghavan *et al.* 2005). Algorithms for stabilization run inside both hardware devices, such as camcorders and still cameras, and software packages for improving the visual quality of shaky videos.

In their paper on full-frame video stabilization, Matsushita, Ofek, Ge *et al.* (2006) give a nice overview of the three major stages of stabilization, namely motion estimation, motion smoothing, and image warping. Motion estimation algorithms often use a similarity transform to handle camera translations, rotations, and zooming. The tricky part is getting these algorithms to lock onto the background motion, which is a result of the camera movement, without getting distracted by independent moving foreground objects. Motion smoothing algorithms recover the low-frequency (slowly varying) part of the motion and then estimate the high-frequency shake component that needs to be removed. Finally, image warping algorithms apply the high-frequency correction to render the original frames as if the camera had undergone only the smooth motion.

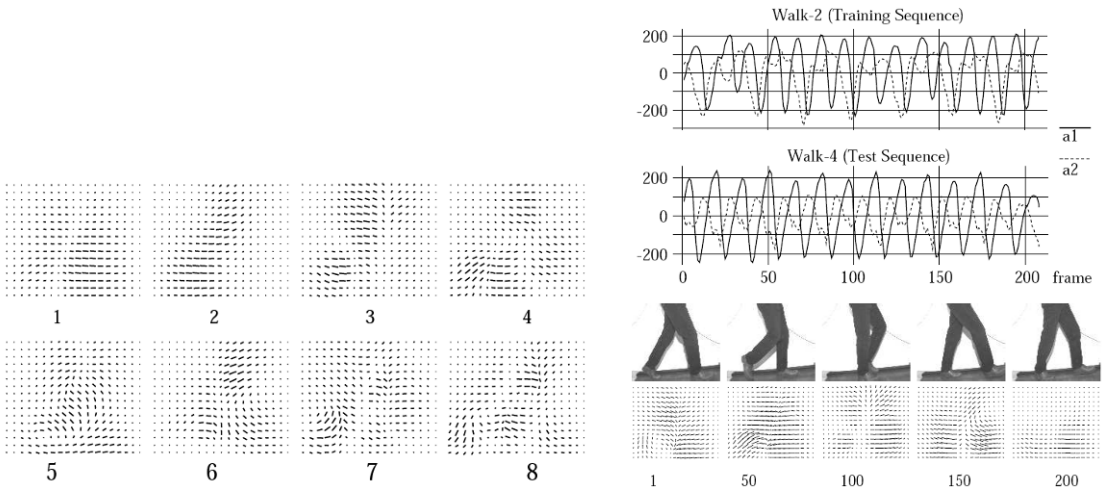
The resulting stabilization algorithms can greatly improve the appearance of shaky videos but they often still contain visual artifacts. For example, image warping can result in missing borders around the image, which must be cropped, filled using information from other frames, or hallucinated using inpainting techniques (Section 10.5.1). Furthermore, video frames captured during fast motion are often blurry. Their appearance can be improved either using deblurring techniques (Section 10.3) or stealing sharper pixels from other frames with less motion or better focus (Matsushita, Ofek, Ge *et al.* 2006). Exercise 8.3 has you implement and test some of these ideas.

In situations where the camera is translating a lot in 3D, e.g., when the videographer is walking, an even better approach is to compute a full structure from motion reconstruction of the camera motion and 3D scene. A smooth 3D camera path can then be computed and the original video re-rendered using view interpolation with the interpolated 3D point cloud serving as the proxy geometry while preserving salient features (Liu, Gleicher, Jin *et al.* 2009). If you have access to a camera array instead of a single video camera, you can do even better using a light field rendering approach (Section 13.3) (Smith, Zhang, Jin *et al.* 2009).

8.2.2 Learned motion models

An alternative to parameterizing the motion field with a geometric deformation such as an affine transform is to learn a set of basis functions tailored to a particular application (Black, Yacoob, Jepson *et al.* 1997). First, a set of dense motion fields (Section 8.4) is computed from a set of training videos. Next, singular value decomposition (SVD) is applied to the stack of motion fields $\boldsymbol{u}_t(\boldsymbol{x})$ to compute the first few singular vectors $\boldsymbol{v}_k(\boldsymbol{x})$. Finally, for a new test sequence, a novel flow field is computed using a coarse-to-fine algorithm that estimates the unknown coefficient a_k in the parameterized flow field

$$\boldsymbol{u}(\boldsymbol{x}) = \sum_k a_k \boldsymbol{v}_k(\boldsymbol{x}). \tag{8.66}$$



(a) (b)

Figure 8.6 Learned parameterized motion fields for a walking sequence (Black, Yacoob, Jepson *et al.* 1997) © 1997 IEEE: (a) learned basis flow fields; (b) plots of motion coefficients over time and corresponding estimated motion fields.

Figure 8.6a shows a set of basis fields learned by observing videos of walking motions. Figure 8.6b shows the temporal evolution of the basis coefficients as well as a few of the recovered parametric motion fields. Note that similar ideas can also be applied to feature tracks (Torresani, Hertzmann, and Bregler 2008), which is a topic we discuss in more detail in Sections 4.1.4 and 12.6.4.

8.3 Spline-based motion

While parametric motion models are useful in a wide variety of applications (such as video stabilization and mapping onto planar surfaces), most image motion is too complicated to be captured by such low-dimensional models.

Traditionally, optical flow algorithms (Section 8.4) compute an independent motion estimate for each pixel, i.e., the number of flow vectors computed is equal to the number of input pixels. The general optical flow analog to Equation (8.1) can thus be written as

$$E_{\mathrm{SSD-OF}}(\{\boldsymbol{u}_i\}) = \sum_i [I_1(\boldsymbol{x}_i + \boldsymbol{u}_i) - I_0(\boldsymbol{x}_i)]^2. \tag{8.67}$$

Notice how in the above equation, the number of variables $\{\boldsymbol{u}_i\}$ is twice the number of measurements, so the problem is underconstrained.

The two classic approaches to this problem, which we study in Section 8.4, are to perform the summation over overlapping regions (the *patch-based* or *window-based* approach) or to add smoothness terms on the $\{\boldsymbol{u}_i\}$ field using *regularization* or *Markov random fields* (Section 3.7). In this section, we describe an alternative approach that lies somewhere between general optical flow (independent flow at each pixel) and parametric flow (a small number of global parameters). The approach is to represent the motion field as a two-dimensional *spline*

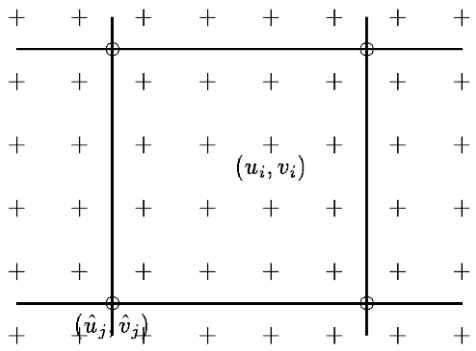


Figure 8.7 Spline motion field: the displacement vectors $\boldsymbol{u}_i = (u_i, v_i)$ are shown as pluses (+) and are controlled by the smaller number of control vertices $\hat{\boldsymbol{u}}_j = (\hat{u}_i, \hat{v}_j)$, which are shown as circles (∘).

controlled by a smaller number of *control vertices* $\{\hat{\boldsymbol{u}}_j\}$ (Figure 8.7),

$$\boldsymbol{u}_i = \sum_j \hat{\boldsymbol{u}}_j B_j(\boldsymbol{x}_i) = \sum_j \hat{\boldsymbol{u}}_j w_{i,j}, \tag{8.68}$$

where the $B_j(\boldsymbol{x}_i)$ are called the *basis functions* and are only non-zero over a small *finite support* interval (Szeliski and Coughlan 1997). We call the $w_{ij} = B_j(\boldsymbol{x}_i)$ *weights* to emphasize that the $\{\boldsymbol{u}_i\}$ are known linear combinations of the $\{\hat{\boldsymbol{u}}_j\}$. Some commonly used spline basis functions are shown in Figure 8.8.

Substituting the formula for the individual per-pixel flow vectors $\boldsymbol{u}_i$ (8.68) into the SSD error metric (8.67) yields a parametric motion formula similar to Equation (8.50). The biggest difference is that the Jacobian $\boldsymbol{J}_1(\boldsymbol{x}'_i)$ (8.52) now consists of the sparse entries in the weight matrix $\boldsymbol{W} = [w_{ij}]$.

In situations where we know something more about the motion field, e.g., when the motion is due to a camera moving in a static scene, we can use more specialized motion models. For example, the *plane plus parallax* model (Section 2.1.5) can be naturally combined with a spline-based motion representation, where the in-plane motion is represented by a homography (6.19) and the out-of-plane parallax d is represented by a scalar variable at each spline control point (Szeliski and Kang 1995; Szeliski and Coughlan 1997).

In many cases, the small number of spline vertices results in a motion estimation problem that is well conditioned. However, if large textureless regions (or elongated edges subject to the aperture problem) persist across several spline patches, it may be necessary to add a *regularization* term to make the problem well posed (Section 3.7.1). The simplest way to do this is to directly add squared difference penalties between adjacent vertices in the spline control mesh $\{\hat{\boldsymbol{u}}_j\}$, as in (3.100). If a multi-resolution (coarse-to-fine) strategy is being used, it is important to re-scale these smoothness terms while going from level to level.

The linear system corresponding to the spline-based motion estimator is sparse and regular. Because it is usually of moderate size, it can often be solved using direct techniques such as Cholesky decomposition (Appendix A.4). Alternatively, if the problem becomes too large and subject to excessive fill-in, iterative techniques such as hierarchically preconditioned conjugate gradient (Szeliski 1990b, 2006b) can be used instead (Appendix A.5).

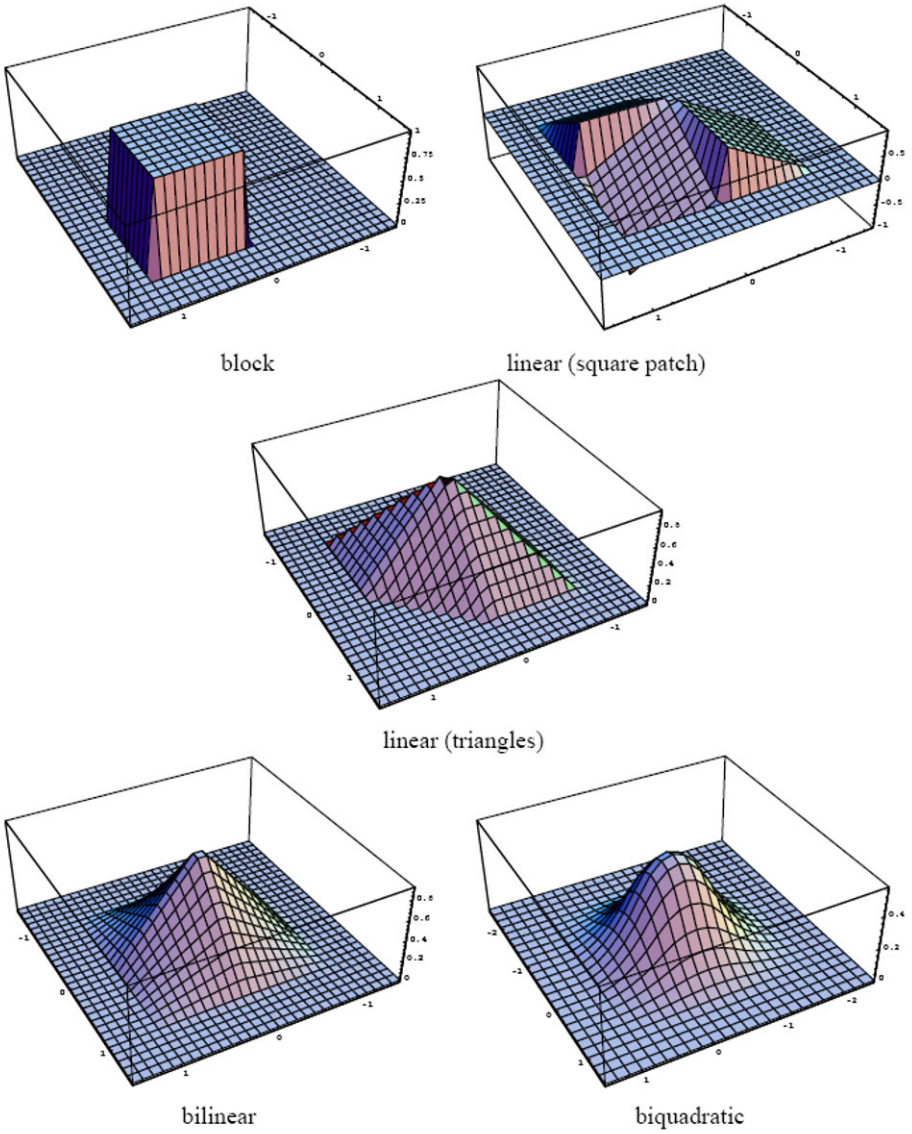
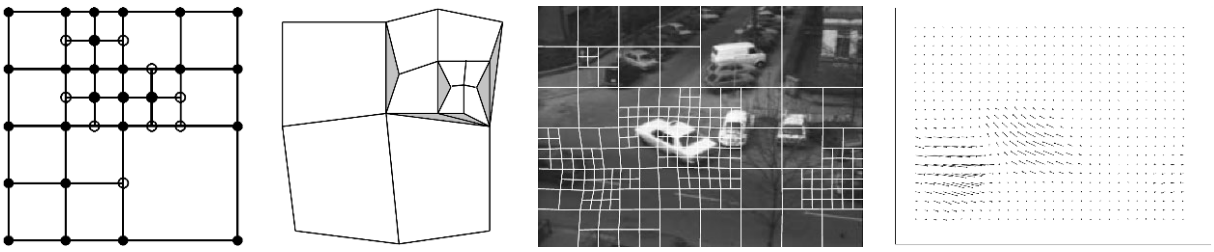


Figure 8.8 Sample spline basis functions (Szeliski and Coughlan 1997) © 1997 Springer. The block (constant) interpolator/basis corresponds to block-based motion estimation (Le Gall 1991). See Section 3.5.1 for more details on spline functions.



(a) (b) (c) (d)

Figure 8.9 Quadtree spline-based motion estimation (Szeliski and Shum 1996) © 1996 IEEE: (a) quadtree spline representation, (b) which can lead to *cracks*, unless the white nodes are constrained to depend on their parents; (c) deformed quadtree spline mesh overlaid on grayscale image; (d) flow field visualized as a needle diagram.

Because of its robustness, spline-based motion estimation has been used for a number of applications, including visual effects (Roble 1999) and medical image registration (Section 8.3.1) (Szeliski and Lavallée 1996; Kybic and Unser 2003).

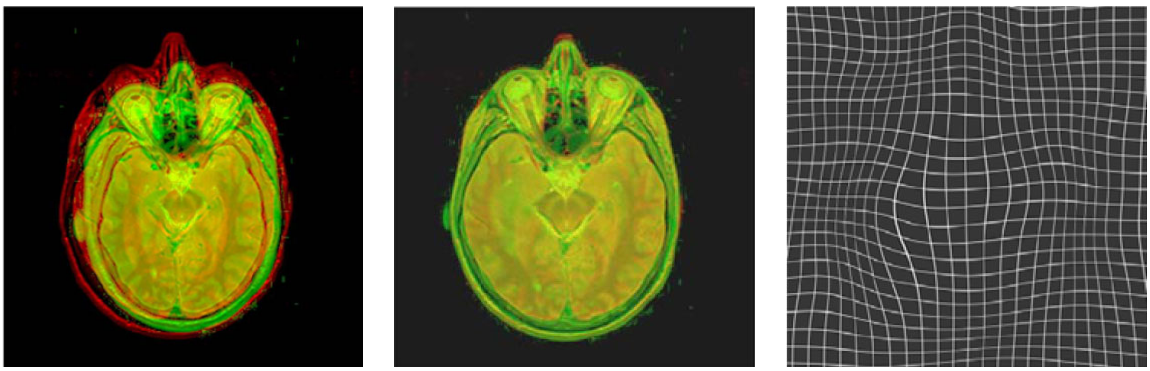
One disadvantage of the basic technique, however, is that the model does a poor job near motion discontinuities, unless an excessive number of nodes is used. To remedy this situation, Szeliski and Shum (1996) propose using a *quadtree* representation embedded in the spline control grid (Figure 8.9a). Large cells are used to present regions of smooth motion, while smaller cells are added in regions of motion discontinuities (Figure 8.9c).

To estimate the motion, a coarse-to-fine strategy is used. Starting with a regular spline imposed over a lower-resolution image, an initial motion estimate is obtained. Spline patches where the motion is inconsistent, i.e., the squared residual (8.67) is above a threshold, are subdivided into smaller patches. In order to avoid *cracks* in the resulting motion field (Figure 8.9b), the values of certain nodes in the refined mesh, i.e., those adjacent to larger cells, need to be *restricted* so that they depend on their parent values. This is most easily accomplished using a hierarchical basis representation for the quadtree spline (Szeliski 1990b) and selectively setting some of the hierarchical basis functions to 0, as described in (Szeliski and Shum 1996).

8.3.1 *Application*: Medical image registration

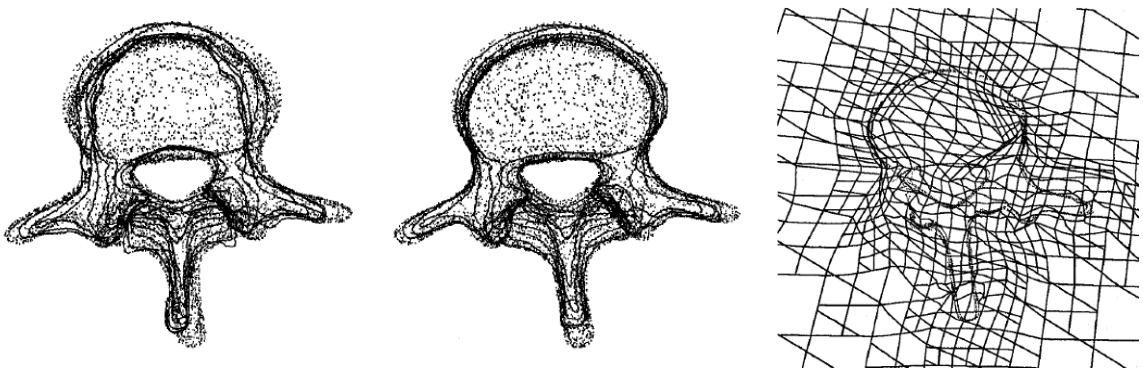
Because they excel at representing smooth *elastic* deformation fields, spline-based motion models have found widespread use in medical image registration (Bajcsy and Kovacic 1989; Szeliski and Lavallée 1996; Christensen, Joshi, and Miller 1997).[10] Registration techniques can be used both to track an individual patient's development or progress over time (a *longitudinal* study) or to match different patient images together to find commonalities and detect variations or pathologies (*cross-sectional* studies). When different imaging *modalities* are being registered, e.g., computed tomography (CT) scans and magnetic resonance images (MRI), *mutual information* measures of similarity are often necessary (Viola and Wells III 1997; Maes, Collignon, Vandermeulen *et al.* 1997).

[10] In computer graphics, such elastic volumetric deformation are known as *free-form deformations* (Sederberg and Parry 1986; Coquillart 1990; Celniker and Gossard 1991).



(a) (b) (c)

Figure 8.10 Elastic brain registration (Kybic and Unser 2003) © 2003 IEEE: (a) original brain atlas and patient MRI images overlaid in red–green; (b) after elastic registration with eight user-specified landmarks (not shown); (c) a cubic B-spline deformation field, shown as a deformed grid.



(a) (b) (c)

Figure 8.11 Octree spline-based image registration of two vertebral surface models (Szeliski and Lavallée 1996) © 1996 Springer: (a) after initial rigid alignment; (b) after elastic alignment; (c) a cross-section through the adapted octree spline deformation field.

Kybic and Unser (2003) provide a nice literature review and describe a complete working system based on representing both the images and the deformation fields as multi-resolution splines. Figure 8.10 shows an example of the Kybic and Unser system being used to register a patient's brain MRI with a labeled brain atlas image. The system can be run in a fully automatic mode but more accurate results can be obtained by locating a few key *landmarks*. More recent papers on deformable medical image registration, including performance evaluations, include (Klein, Staring, and Pluim 2007; Glocker, Komodakis, Tziritas *et al.* 2008).

As with other applications, regular volumetric splines can be enhanced using selective refinement. In the case of 3D volumetric image or surface registration, these are known as *octree splines* (Szeliski and Lavallée 1996) and have been used to register medical surface models such as vertebrae and faces from different patients (Figure 8.11).

8.4 Optical flow

The most general (and challenging) version of motion estimation is to compute an independent estimate of motion at *each* pixel, which is generally known as *optical* (or *optic*) *flow*. As we mentioned in the previous section, this generally involves minimizing the brightness or color difference between corresponding pixels summed over the image,

$$E_{\mathrm{SSD-OF}}(\{\boldsymbol{u}_i\}) = \sum_i [I_1(\boldsymbol{x}_i + \boldsymbol{u}_i) - I_0(\boldsymbol{x}_i)]^2. \tag{8.69}$$

Since the number of variables $\{\boldsymbol{u}_i\}$ is twice the number of measurements, the problem is underconstrained. The two classic approaches to this problem are to perform the summation *locally* over overlapping regions (the *patch-based* or *window-based* approach) or to add smoothness terms on the $\{\boldsymbol{u}_i\}$ field using regularization or Markov random fields (Section 3.7) and to search for a global minimum.

The patch-based approach usually involves using a Taylor series expansion of the displaced image function (8.35) in order to obtain sub-pixel estimates (Lucas and Kanade 1981). Anandan (1989) shows how a series of local discrete search steps can be interleaved with Lucas–Kanade incremental refinement steps in a coarse-to-fine pyramid scheme, which allows the estimation of large motions, as described in Section 8.1.1. He also analyzes how the *uncertainty* in local motion estimates is related to the eigenvalues of the local Hessian matrix $\boldsymbol{A}_i$ (8.44), as shown in Figures 8.3–8.4.

Bergen, Anandan, Hanna *et al.* (1992) develop a unified framework for describing both parametric (Section 8.2) and patch-based optic flow algorithms and provide a nice introduction to this topic. After each iteration of optic flow estimation in a coarse-to-fine pyramid, they re-warp one of the images so that only incremental flow estimates are computed (Section 8.1.1). When overlapping patches are used, an efficient implementation is to first compute the outer products of the gradients and intensity errors (8.40–8.41) at every pixel and then perform the overlapping window sums using a moving average filter.[11]

Instead of solving for each motion (or motion update) independently, Horn and Schunck (1981) develop a regularization-based framework where (8.69) is simultaneously minimized over all flow vectors $\{\boldsymbol{u}_i\}$. In order to constrain the problem, smoothness constraints, i.e., squared penalties on flow derivatives, are added to the basic per-pixel error metric. Because the technique was originally developed for small motions in a variational (continuous function) framework, the linearized *brightness constancy constraint* corresponding to (8.35), i.e., (8.38), is more commonly written as an analytic integral

$$E_{\mathrm{HS}} = \int (I_x u + I_y v + I_t)^2 \, dx \, dy, \tag{8.70}$$

where $(I_x, I_y) = \nabla I_1 = \boldsymbol{J}_1$ and $I_t = e_i$ is the *temporal derivative*, i.e., the brightness change between images. The Horn and Schunck model can also be viewed as the limiting case of spline-based motion estimation as the splines become 1x1 pixel patches.

It is also possible to combine ideas from local and global flow estimation into a single framework by using a locally aggregated (as opposed to single-pixel) Hessian as the brightness constancy term (Bruhn, Weickert, and Schnörr 2005). Consider the discrete analog

[11] Other smoothing or aggregation filters can also be used at this stage (Bruhn, Weickert, and Schnörr 2005).

(8.35) to the analytic global energy (8.70),

$$E_{\mathrm{HSD}} = \sum_i \boldsymbol{u}_i^T [\boldsymbol{J}_i \boldsymbol{J}_i^T] \boldsymbol{u}_i + 2 e_i \boldsymbol{J}_i^T \boldsymbol{u}_i + e_i^2. \tag{8.71}$$

If we replace the per-pixel (rank 1) Hessians $\boldsymbol{A}_i = [\boldsymbol{J}_i \boldsymbol{J}_i^T]$ and residuals $\boldsymbol{b}_i = \boldsymbol{J}_i e_i$ with area-aggregated versions (8.40–8.41), we obtain a global minimization algorithm where region-based brightness constraints are used.

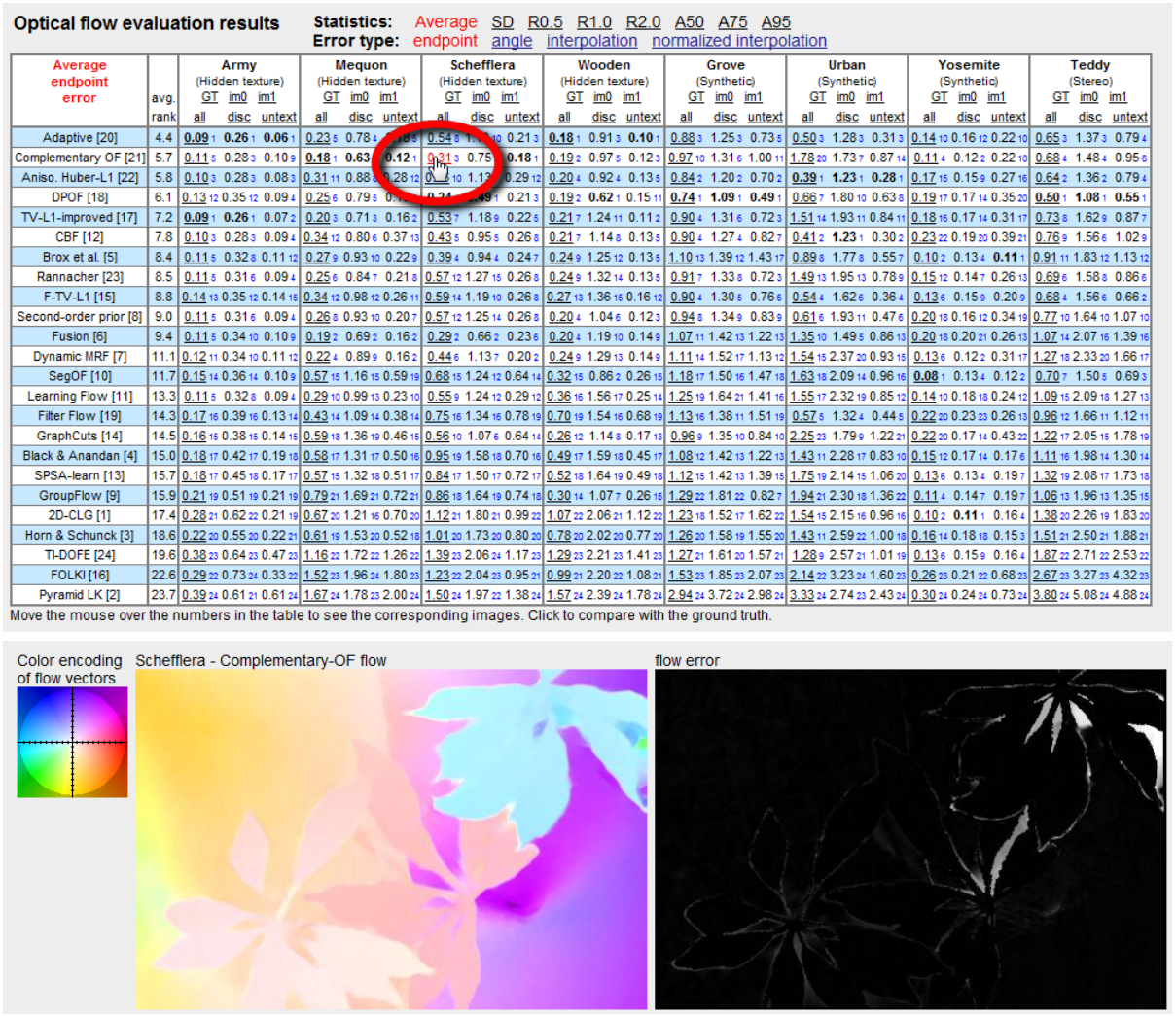
Another extension to the basic optic flow model is to use a combination of global (parametric) and local motion models. For example, if we know that the motion is due to a camera moving in a static scene (rigid motion), we can re-formulate the problem as the estimation of a per-pixel depth along with the parameters of the global camera motion (Adiv 1989; Hanna 1991; Bergen, Anandan, Hanna *et al.* 1992; Szeliski and Coughlan 1997; Nir, Bruckstein, and Kimmel 2008; Wedel, Cremers, Pock *et al.* 2009). Such techniques are closely related to stereo matching (Chapter 11). Alternatively, we can estimate either per-image or per-segment affine motion models combined with per-pixel *residual* corrections (Black and Jepson 1996; Ju, Black, and Jepson 1996; Chang, Tekalp, and Sezan 1997; Mémin and Pérez 2002). We revisit this topic in Section 8.5.

Of course, image brightness may not always be an appropriate metric for measuring appearance consistency, e.g., when the lighting in an image is varying. As discussed in Section 8.1, matching gradients, filtered images, or other metrics such as image Hessians (second derivative measures) may be more appropriate. It is also possible to locally compute the *phase* of steerable filters in the image, which is insensitive to both bias and gain transformations (Fleet and Jepson 1990). Papenberg, Bruhn, Brox *et al.* (2006) review and explore such constraints and also provide a detailed analysis and justification for iteratively re-warping images during incremental flow computation.

Because the brightness constancy constraint is evaluated at each pixel independently, rather than being summed over patches where the constant flow assumption may be violated, global optimization approaches tend to perform better near motion discontinuities. This is especially true if robust metrics are used in the smoothness constraint (Black and Anandan 1996; Bab-Hadiashar and Suter 1998a).[12] One popular choice for robust metrics in the L_1 norm, also known as *total variation* (TV), which results in a convex energy whose global minimum can be found (Bruhn, Weickert, and Schnörr 2005; Papenberg, Bruhn, Brox *et al.* 2006). Anisotropic smoothness priors, which apply a different smoothness in the directions parallel and perpendicular to the image gradient, are another popular choice (Nagel and Enkelmann 1986; Sun, Roth, Lewis *et al.* 2008; Werlberger, Trobin, Pock *et al.* 2009). It is also possible to learn a set of better smoothness constraints (derivative filters and robust functions) from a set of paired flow and intensity images (Sun, Roth, Lewis *et al.* 2008). Additional details on some of these techniques are given by Baker, Black, Lewis *et al.* (2007) and Baker, Scharstein, Lewis *et al.* (2009).

Because of the large, two-dimensional search space in estimating flow, most algorithms use variations of gradient descent and coarse-to-fine continuation methods to minimize the global energy function. This contrasts starkly with stereo matching (which is an "easier" one-dimensional disparity estimation problem), where combinatorial optimization techniques have been the method of choice for the last decade.

[12] Robust brightness metrics (Section 8.1, (8.2)) can also help improve the performance of window-based approaches (Black and Anandan 1996).



Optical flow evaluation results

Statistics: Average SD R0.5 R1.0 R2.0 A50 A75 A95
Error type: endpoint angle interpolation normalized interpolation

Average endpoint error	avg. rank	Army (Hidden texture) GT im0 im1 all disc untext	Mequon (Hidden texture) GT im0 im1 all disc untext	Schefflera (Hidden texture) GT im0 im1 all disc untext	Wooden (Hidden texture) GT im0 im1 all disc untext	Grove (Synthetic) GT im0 im1 all disc untext	Urban (Synthetic) GT im0 im1 all disc untext	Yosemite (Synthetic) GT im0 im1 all disc untext	Teddy (Stereo) GT im0 im1 all disc untext
Adaptive [20]	4.4	**0.09** 1 **0.26** 1 **0.06** 1	0.23 8 0.78 4 0.18 8	0.54 8 1.51 10 0.21 3	**0.18** 1 0.91 3 **0.10** 1	0.88 3 1.25 3 0.73 5	0.50 3 1.28 3 0.31 3	0.14 10 0.16 12 0.22 10	0.65 3 1.37 3 0.79 4
Complementary OF [21]	5.7	0.11 5 0.28 3 0.10 9	**0.18** 1 **0.63** 1 **0.12** 1	0.31 3 0.75 1 **0.18** 1	0.19 2 0.97 5 0.12 3	0.97 10 1.31 6 1.00 11	1.78 20 1.73 7 0.87 14	0.11 4 0.12 2 0.22 10	0.68 4 1.48 4 0.95 8
Aniso. Huber-L1 [22]	5.8	0.10 3 0.28 3 0.08 3	0.31 11 0.88 [illegible] 0.28 12	0.[illegible] 10 1.13 [illegible] 0.29 12	0.20 4 0.92 4 0.13 5	0.84 2 1.20 2 0.70 2	**0.39** 1 **1.23** 1 **0.28** 1	0.17 15 0.15 9 0.27 16	0.64 2 1.36 2 0.79 4
DPOF [18]	6.1	0.13 12 0.35 12 0.09 4	0.25 6 0.79 5 0.[illegible]	**0.[illegible]** 1 [illegible] 0.21 3	0.19 2 **0.62** 1 0.15 11	**0.74** 1 **1.09** 1 **0.49** 1	0.66 7 1.80 10 0.63 8	0.19 17 0.17 14 0.35 20	**0.50** 1 **1.08** 1 **0.55** 1
TV-L1-improved [17]	7.2	**0.09** 1 **0.26** 1 0.07 2	0.20 3 0.71 3 0.16 2	0.53 7 1.18 9 0.22 5	0.21 7 1.24 11 0.11 2	0.90 4 1.31 6 0.72 3	1.51 14 1.93 11 0.84 11	0.18 16 0.17 14 0.31 17	0.73 8 1.62 9 0.87 7
CBF [12]	7.8	0.10 3 0.28 3 0.09 4	0.34 12 0.80 6 0.37 13	0.43 5 0.95 5 0.26 8	0.21 7 1.14 8 0.13 5	0.90 4 1.27 4 0.82 7	0.41 2 **1.23** 1 0.30 2	0.23 22 0.19 20 0.39 21	0.76 9 1.56 6 1.02 9
Brox et al. [5]	8.4	0.11 5 0.32 8 0.11 12	0.27 9 0.93 10 0.22 9	0.39 4 0.94 4 0.24 7	0.24 9 1.25 12 0.13 5	1.10 13 1.39 12 1.43 17	0.89 8 1.77 8 0.55 7	0.10 2 0.13 4 **0.11** 1	0.91 11 1.83 12 1.13 12
Rannacher [23]	8.5	0.11 5 0.31 6 0.09 4	0.25 6 0.84 7 0.21 8	0.57 12 1.27 15 0.26 8	0.24 9 1.32 14 0.13 5	0.91 7 1.33 8 0.72 3	1.49 13 1.95 13 0.78 9	0.15 12 0.14 7 0.26 13	0.69 6 1.58 8 0.86 6
F-TV-L1 [15]	8.8	0.14 13 0.35 12 0.14 15	0.34 12 0.98 12 0.26 11	0.59 14 1.19 10 0.26 8	0.27 13 1.36 15 0.16 12	0.90 4 1.30 5 0.76 6	0.54 4 1.62 6 0.36 4	0.13 6 0.15 9 0.20 9	0.68 4 1.56 6 0.66 2
Second-order prior [8]	9.0	0.11 5 0.31 6 0.09 4	0.26 8 0.93 10 0.20 7	0.57 12 1.25 14 0.26 8	0.20 4 1.04 6 0.12 3	0.94 8 1.34 9 0.83 9	0.61 6 1.93 11 0.47 6	0.20 18 0.16 12 0.34 19	0.77 10 1.64 10 1.07 10
Fusion [6]	9.4	0.11 5 0.34 10 0.10 9	0.19 2 0.69 2 0.16 2	0.29 2 0.66 2 0.23 6	0.20 4 1.19 10 0.14 9	1.07 11 1.42 13 1.22 13	1.35 10 1.49 5 0.86 13	0.20 18 0.20 21 0.26 13	1.07 14 2.07 16 1.39 16
Dynamic MRF [7]	11.1	0.12 11 0.34 10 0.11 12	0.22 4 0.89 9 0.16 2	0.44 6 1.13 7 0.20 2	0.24 9 1.29 13 0.14 9	1.11 14 1.52 17 1.13 12	1.54 15 2.37 20 0.93 15	0.13 6 0.12 2 0.31 17	1.27 18 2.33 20 1.66 17
SegOF [10]	11.7	0.15 14 0.36 14 0.10 9	0.57 15 1.16 15 0.59 19	0.68 15 1.24 12 0.64 14	0.32 15 0.86 2 0.26 15	1.18 17 1.50 16 1.47 18	1.63 18 2.09 14 0.96 16	**0.08** 1 0.13 4 0.12 2	0.70 7 1.50 5 0.69 3
Learning Flow [11]	13.3	0.11 5 0.32 8 0.09 4	0.29 10 0.99 13 0.23 10	0.55 9 1.24 12 0.29 12	0.36 16 1.56 17 0.25 14	1.25 19 1.64 21 1.41 16	1.55 17 2.32 19 0.85 12	0.14 10 0.18 18 0.24 12	1.09 15 2.09 18 1.27 13
Filter Flow [19]	14.3	0.17 16 0.39 16 0.13 14	0.43 14 1.09 14 0.38 14	0.75 16 1.34 16 0.78 19	0.70 19 1.54 16 0.68 19	1.13 16 1.38 11 1.51 19	0.57 5 1.32 4 0.44 5	0.22 20 0.23 23 0.26 13	0.96 12 1.66 11 1.12 11
GraphCuts [14]	14.5	0.16 15 0.38 15 0.14 15	0.59 18 1.36 19 0.46 15	0.56 10 1.07 6 0.64 14	0.26 12 1.14 8 0.17 13	0.96 9 1.35 10 0.84 10	2.25 23 1.79 9 1.22 21	0.22 20 0.17 14 0.43 22	1.22 17 2.05 15 1.78 19
Black & Anandan [4]	15.0	0.18 17 0.42 17 0.19 18	0.58 17 1.31 17 0.50 16	0.95 19 1.58 18 0.70 16	0.49 17 1.59 18 0.45 17	1.08 12 1.42 13 1.22 13	1.43 11 2.28 17 0.83 10	0.15 12 0.17 14 0.17 6	1.11 16 1.98 14 1.30 14
SPSA-learn [13]	15.7	0.18 17 0.45 18 0.17 17	0.57 15 1.32 18 0.51 17	0.84 17 1.50 17 0.72 17	0.52 18 1.64 19 0.49 18	1.12 15 1.42 13 1.39 15	1.75 19 2.14 15 1.06 20	0.13 6 0.13 4 0.19 7	1.32 19 2.08 17 1.73 18
GroupFlow [9]	15.9	0.21 19 0.51 19 0.21 19	0.79 21 1.69 21 0.72 21	0.86 18 1.64 19 0.74 18	0.30 14 1.07 7 0.26 15	1.29 22 1.81 22 0.82 7	1.94 21 2.30 18 1.36 22	0.11 4 0.14 7 0.19 7	1.06 13 1.96 13 1.35 15
2D-CLG [1]	17.4	0.28 21 0.62 22 0.21 19	0.67 20 1.21 16 0.70 20	1.12 21 1.80 21 0.99 22	1.07 22 2.06 21 1.12 22	1.23 18 1.52 17 1.62 22	1.54 15 2.15 16 0.96 16	0.10 2 **0.11** 1 0.16 4	1.38 20 2.26 19 1.83 20
Horn & Schunck [3]	18.6	0.22 20 0.55 20 0.22 21	0.61 19 1.53 20 0.52 18	1.01 20 1.73 20 0.80 20	0.78 20 2.02 20 0.77 20	1.26 20 1.58 19 1.55 20	1.43 11 2.59 22 1.00 18	0.16 14 0.18 18 0.15 3	1.51 21 2.50 21 1.88 21
TI-DOFE [24]	19.6	0.38 23 0.64 23 0.47 23	1.16 22 1.72 22 1.26 22	1.39 23 2.06 24 1.17 23	1.29 23 2.21 23 1.41 23	1.27 21 1.61 20 1.57 21	1.28 9 2.57 21 1.01 19	0.13 6 0.15 9 0.16 4	1.87 22 2.71 22 2.53 22
FOLKI [16]	22.6	0.29 22 0.73 24 0.33 22	1.52 23 1.96 24 1.80 23	1.23 22 2.04 23 0.95 21	0.99 21 2.20 22 1.08 21	1.53 23 1.85 23 2.07 23	2.14 22 3.23 24 1.60 23	0.26 23 0.21 22 0.68 23	2.67 23 3.27 23 4.32 23
Pyramid LK [2]	23.7	0.39 24 0.61 21 0.61 24	1.67 24 1.78 23 2.00 24	1.50 24 1.97 22 1.38 24	1.57 24 2.39 24 1.78 24	2.94 24 3.72 24 2.98 24	3.33 24 2.74 23 2.43 24	0.30 24 0.24 24 0.73 24	3.80 24 5.08 24 4.88 24

Move the mouse over the numbers in the table to see the corresponding images. Click to compare with the ground truth.

Figure 8.12 Evaluation of the results of 24 optical flow algorithms, October 2009, http://vision.middlebury.edu/flow/, (Baker, Scharstein, Lewis *et al.* 2009). By moving the mouse pointer over an underlined performance score, the user can interactively view the corresponding flow and error maps. Clicking on a score toggles between the computed and ground truth flows. Next to each score, the corresponding rank in the current column is indicated by a smaller blue number. The minimum (best) score in each column is shown in boldface. The table is sorted by the average rank (computed over all 24 columns, three region masks for each of the eight sequences). The average rank serves as an *approximate* measure of performance *under the selected metric/statistic*.

Fortunately, combinatorial optimization methods based on Markov random fields are beginning to appear and tend to be among the better-performing methods on the recently released optical flow database (Baker, Black, Lewis *et al.* 2007).[13]

Examples of such techniques include the one developed by Glocker, Paragios, Komodakis *et al.* (2008), who use a coarse-to-fine strategy with per-pixel 2D uncertainty estimates, which are then used to guide the refinement and search at the next finer level. Instead of using gradient descent to refine the flow estimates, a combinatorial search over discrete displacement labels (which is able to find better energy minima) is performed using their Fast-PD algorithm (Komodakis, Tziritas, and Paragios 2008).

Lempitsky, Roth, and Rother. (2008) use fusion moves (Lempitsky, Rother, and Blake 2007) over proposals generated from basic flow algorithms (Horn and Schunck 1981; Lucas and Kanade 1981) to find good solutions. The basic idea behind fusion moves is to replace portions of the current best estimate with hypotheses generated by more basic techniques (or their shifted versions) and to alternate them with local gradient descent for better energy minimization.

The field of accurate motion estimation continues to evolve at a rapid pace, with significant advances in performance occurring every year. The optical flow evaluation Web site (http://vision.middlebury.edu/flow/) is a good source of pointers to high-performing recently developed algorithms (Figure 8.12).

8.4.1 Multi-frame motion estimation

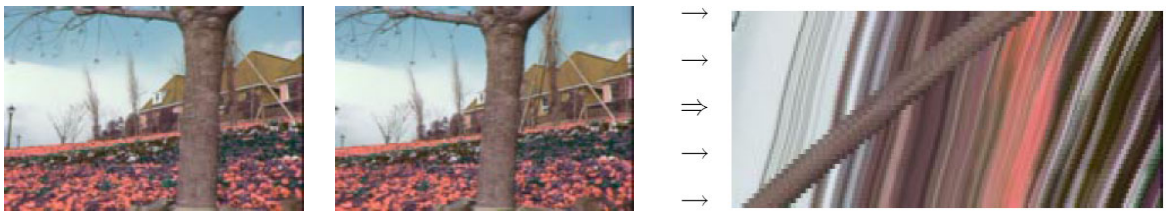
So far, we have looked at motion estimation as a two-frame problem, where the goal is to compute a motion field that aligns pixels from one image with those in another. In practice, motion estimation is usually applied to video, where a whole sequence of frames is available to perform this task.

One classic approach to multi-frame motion is to *filter* the spatio-temporal volume using oriented or steerable filters (Heeger 1988), in a manner analogous to oriented edge detection (Section 3.2.3). Figure 8.13 shows two frames from the commonly used *flower garden* sequence, as well as a horizontal slice through the spatio-temporal volume, i.e., the 3D volume created by stacking all of the video frames together. Because the pixel motion is mostly horizontal, the slopes of individual (textured) pixel tracks, which correspond to their horizontal velocities, can clearly be seen. Spatio-temporal filtering uses a 3D volume around each pixel to determine the best orientation in space–time, which corresponds directly to a pixel's velocity.

Unfortunately, in order to obtain reasonably accurate velocity estimates everywhere in an image, spatio-temporal filters have moderately large extents, which severely degrades the quality of their estimates near motion discontinuities. (This same problem is endemic in 2D window-based motion estimators.) An alternative to full spatio-temporal filtering is to estimate more local spatio-temporal derivatives and use them inside a global optimization framework to fill in textureless regions (Bruhn, Weickert, and Schnörr 2005; Govindu 2006).

Another alternative is to simultaneously estimate multiple motion estimates, while also optionally reasoning about occlusion relationships (Szeliski 1999). Figure 8.13c shows schematically one potential approach to this problem. The horizontal arrows show the locations of

[13] http://vision.middlebury.edu/flow/.



(a) (b) (c)

Figure 8.13 Slice through a spatio-temporal volume (Szeliski 1999) © 1999 IEEE: (a–b) two frames from the *flower garden* sequence; (c) a horizontal slice through the complete spatio-temporal volume, with the arrows indicating locations of potential key frames where flow is estimated. Note that the colors for the flower garden sequence are incorrect; the correct colors (yellow flowers) are shown in Figure 8.15.

keyframes s where motion is estimated, while other slices indicate video frames t whose colors are matched with those predicted by interpolating between the keyframes. Motion estimation can be cast as a global energy minimization problem that simultaneously minimizes brightness compatibility and flow compatibility terms between keyframes and other frames, in addition to using robust smoothness terms.

The multi-view framework is potentially even more appropriate for rigid scene motion (multi-view stereo) (Section 11.6), where the unknowns at each pixel are disparities and occlusion relationships can be determined directly from pixel depths (Szeliski 1999; Kolmogorov and Zabih 2002). However, it may also be applicable to general motion, with the addition of models for object accelerations and occlusion relationships.

8.4.2 *Application*: Video denoising

Video denoising is the process of removing noise and other artifacts such as scratches from film and video (Kokaram 2004). Unlike single image denoising, where the only information available is in the current picture, video denoisers can average or borrow information from adjacent frames. However, in order to do this without introducing blur or jitter (irregular motion), they need accurate per-pixel motion estimates.

Exercise 8.7 lists some of the steps required, which include the ability to determine if the current motion estimate is accurate enough to permit averaging with other frames. Gai and Kang (2009) describe their recently developed restoration process, which involves a series of additional steps to deal with the special characteristics of vintage film.

8.4.3 *Application*: De-interlacing

Another commonly used application of per-pixel motion estimation is video de-interlacing, which is the process of converting a video taken with alternating fields of even and odd lines to a non-interlaced signal that contains both fields in each frame (de Haan and Bellers 1998). Two simple de-interlacing techniques are *bob*, which copies the line above or below the missing line from the same field, and *weave*, which copies the corresponding line from the field before or after. The names come from the visual artifacts generated by these two simple techniques: bob introduces an up-and-down bobbing motion along strong horizontal

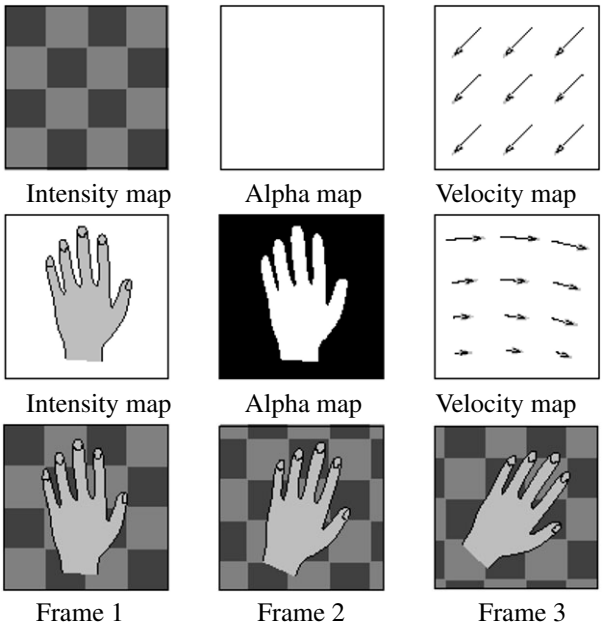


Figure 8.14 Layered motion estimation framework (Wang and Adelson 1994) © 1994 IEEE: The top two rows describe the two layers, each of which consists of an intensity (color) image, an alpha mask (black=transparent), and a parametric motion field. The layers are composited with different amounts of motion to recreate the video sequence.

lines; weave can lead to a "zippering" effect along horizontally translating edges. Replacing these copy operators with averages can help but does not completely remove these artifacts.

A wide variety of improved techniques have been developed for this process, which is often embedded in specialized DSP chips found inside video digitization boards in computers (since broadcast video is often interlaced, while computer monitors are not). A large class of these techniques estimates local per-pixel motions and interpolates the missing data from the information available in spatially and temporally adjacent fields. Dai, Baker, and Kang (2009) review this literature and propose their own algorithm, which selects among seven different interpolation functions at each pixel using an MRF framework.

8.5 Layered motion

In many situation, visual motion is caused by the movement of a small number of objects at different depths in the scene. In such situations, the pixel motions can be described more succinctly (and estimated more reliably) if pixels are grouped into appropriate objects or *layers* (Wang and Adelson 1994).

Figure 8.14 shows this approach schematically. The motion in this sequence is caused by the translational motion of the checkered background and the rotation of the foreground hand. The complete motion sequence can be reconstructed from the appearance of the foreground and background elements, which can be represented as alpha-matted images (*sprites* or *video objects*) and the parametric motion corresponding to each layer. Displacing and compositing

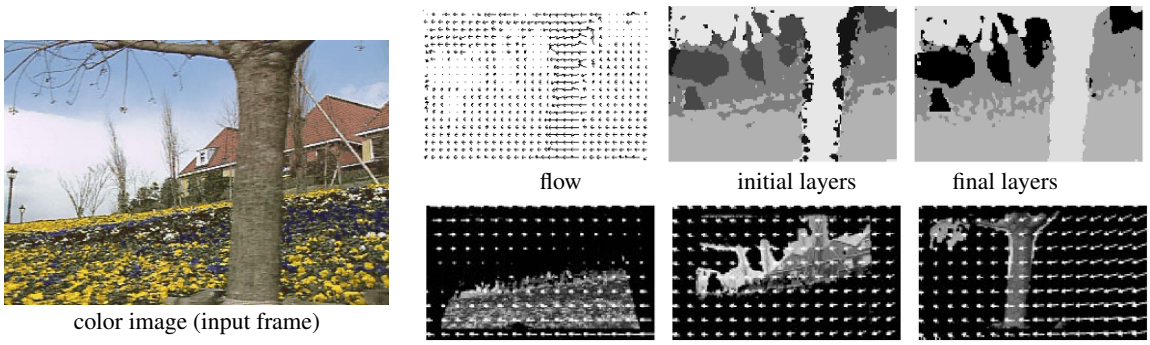


Figure 8.15 Layered motion estimation results (Wang and Adelson 1994) © 1994 IEEE.

these layers in back to front order (Section 3.1.3) recreates the original video sequence.

Layered motion representations not only lead to compact representations (Wang and Adelson 1994; Lee, ge Chen, lung Bruce Lin *et al.* 1997), but they also exploit the information available in multiple video frames, as well as accurately modeling the appearance of pixels near motion discontinuities. This makes them particularly suited as a representation for image-based rendering (Section 13.2.1) (Shade, Gortler, He *et al.* 1998; Zitnick, Kang, Uyttendaele *et al.* 2004) as well as object-level video editing.

To compute a layered representation of a video sequence, Wang and Adelson (1994) first estimate affine motion models over a collection of non-overlapping patches and then cluster these estimates using k-means. They then alternate between assigning pixels to layers and recomputing motion estimates for each layer using the assigned pixels, using a technique first proposed by Darrell and Pentland (1991). Once the parametric motions and pixel-wise layer assignments have been computed for each frame independently, layers are constructed by warping and merging the various layer pieces from all of the frames together. Median filtering is used to produce sharp composite layers that are robust to small intensity variations, as well as to infer occlusion relationships between the layers. Figure 8.15 shows the results of this process on the *flower garden* sequence. You can see both the initial and final layer assignments for one of the frames, as well as the composite flow and the alpha-matted layers with their corresponding flow vectors overlaid.

In follow-on work, Weiss and Adelson (1996) use a formal probabilistic mixture model to infer both the optimal number of layers and the per-pixel layer assignments. Weiss (1997) further generalizes this approach by replacing the per-layer affine motion models with smooth regularized per-pixel motion estimates, which allows the system to better handle curved and undulating layers, such as those seen in most real-world sequences.

The above approaches, however, still make a distinction between estimating the motions and layer assignments and then later estimating the layer colors. In the system described by Baker, Szeliski, and Anandan (1998), the generative model illustrated in Figure 8.14 is generalized to account for real-world rigid motion scenes. The motion of each frame is described using a 3D camera model and the motion of each layer is described using a 3D plane equation plus per-pixel residual depth offsets (the *plane plus parallax* representation (Section 2.1.5)). The initial layer estimation proceeds in a manner similar to that of Wang and Adelson (1994),

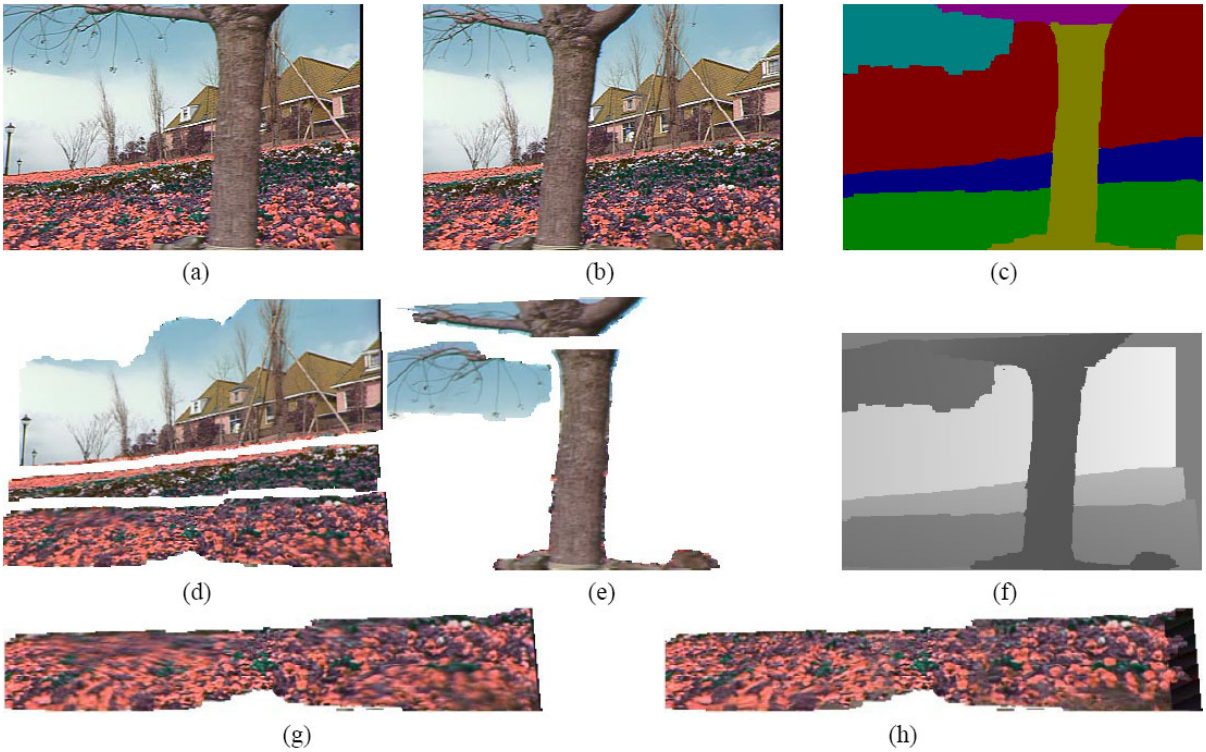


Figure 8.16 Layered stereo reconstruction (Baker, Szeliski, and Anandan 1998) © 1998 IEEE: (a) first and (b) last input images; (c) initial segmentation into six layers; (d) and (e) the six layer sprites; (f) depth map for planar sprites (darker denotes closer); front layer (g) before and (h) after residual depth estimation. Note that the colors for the flower garden sequence are incorrect; the correct colors (yellow flowers) are shown in Figure 8.15.

o

except that rigid planar motions (homographies) are used instead of affine motion models. The final model refinement, however, jointly re-optimizes the layer pixel color and opacity values L_l and the 3D depth, plane, and motion parameters z_l, $\boldsymbol{n}_l$, and $\boldsymbol{P}_t$ by minimizing the discrepancy between the re-synthesized and observed motion sequences (Baker, Szeliski, and Anandan 1998).

Figure 8.16 shows the final results obtained with this algorithm. As you can see, the motion boundaries and layer assignments are much crisper than those in Figure 8.15. Because of the per-pixel depth offsets, the individual layer color values are also sharper than those obtained with affine or planar motion models. While the original system of Baker, Szeliski, and Anandan (1998) required a rough initial assignment of pixels to layers, Torr, Szeliski, and Anandan (2001) describe automated Bayesian techniques for initializing this system and determining the optimal number of layers.

Layered motion estimation continues to be an active area of research. Representative papers in this area include (Sawhney and Ayer 1996; Jojic and Frey 2001; Xiao and Shah 2005; Kumar, Torr, and Zisserman 2008; Thayananthan, Iwasaki, and Cipolla 2008; Schoenemann and Cremers 2008).

Of course, layers are not the only way to introduce segmentation into motion estimation.

A large number of algorithms have been developed that alternate between estimating optic flow vectors and segmenting them into coherent regions (Black and Jepson 1996; Ju, Black, and Jepson 1996; Chang, Tekalp, and Sezan 1997; Mémin and Pérez 2002; Cremers and Soatto 2005). Some of the more recent techniques rely on first segmenting the input color images and then estimating per-segment motions that produce a coherent motion field while also modeling occlusions (Zitnick, Kang, Uyttendaele *et al.* 2004; Zitnick, Jojic, and Kang 2005; Stein, Hoiem, and Hebert 2007; Thayananthan, Iwasaki, and Cipolla 2008).

8.5.1 *Application*: Frame interpolation

Frame interpolation is another widely used application of motion estimation, often implemented in the same circuitry as de-interlacing hardware required to match an incoming video to a monitor's actual refresh rate. As with de-interlacing, information from novel in-between frames needs to be interpolated from preceding and subsequent frames. The best results can be obtained if an accurate motion estimate can be computed at each unknown pixel's location. However, in addition to computing the motion, occlusion information is critical to prevent colors from being contaminated by moving foreground objects that might obscure a particular pixel in a preceding or subsequent frame.

In a little more detail, consider Figure 8.13c and assume that the arrows denote keyframes between which we wish to interpolate additional images. The orientations of the streaks in this figure encode the velocities of individual pixels. If the same motion estimate $\boldsymbol{u}_0$ is obtained at location $\boldsymbol{x}_0$ in image I_0 as is obtained at location $\boldsymbol{x}_0 + \boldsymbol{u}_0$ in image I_1, the flow vectors are said to be *consistent*. This motion estimate can be transferred to location $\boldsymbol{x}_0 + t\boldsymbol{u}_0$ in the image I_t being generated, where $t \in (0, 1)$ is the time of interpolation. The final color value at pixel $\boldsymbol{x}_0 + t\boldsymbol{u}_0$ can be computed as a linear blend,

$$I_t(\boldsymbol{x}_0 + t\boldsymbol{u}_0) = (1 - t)I_0(\boldsymbol{x}_0) + tI_1(\boldsymbol{x}_0 + \boldsymbol{u}_0). \tag{8.72}$$

If, however, the motion vectors are different at corresponding locations, some method must be used to determine which is correct and which image contains colors that are occluded. The actual reasoning is even more subtle than this. One example of such an interpolation algorithm, based on earlier work in depth map interpolation (Shade, Gortler, He *et al.* 1998; Zitnick, Kang, Uyttendaele *et al.* 2004) which is the one used in the flow evaluation paper of Baker, Black, Lewis *et al.* (2007); Baker, Scharstein, Lewis *et al.* (2009). An even higher-quality frame interpolation algorithm, which uses gradient-based reconstruction, is presented by Mahajan, Huang, Matusik *et al.* (2009).

8.5.2 Transparent layers and reflections

A special case of layered motion that occurs quite often is transparent motion, which is usually caused by reflections seen in windows and picture frames (Figures 8.17 and 8.18).

Some of the early work in this area handles transparent motion by either just estimating the component motions (Shizawa and Mase 1991; Bergen, Burt, Hingorani *et al.* 1992; Darrell and Simoncelli 1993; Irani, Rousso, and Peleg 1994) or by assigning individual pixels to competing motion layers (Darrell and Pentland 1995; Black and Anandan 1996; Ju, Black, and Jepson 1996), which is appropriate for scenes partially seen through a fine occluder (e.g., foliage). However, to accurately separate truly transparent layers, a better model for

Figure 8.17 Light reflecting off the transparent glass of a picture frame: (a) first image from the input sequence; (b) dominant motion layer *min-composite*; (c) secondary motion residual layer *max-composite*; (d–e) final estimated picture and reflection layers The original images are from Black and Anandan (1996), while the separated layers are from Szeliski, Avidan, and Anandan (2000) © 2000 IEEE.

motion due to reflections is required. Because of the way that light is both reflected from and transmitted through a glass surface, the correct model for reflections is an *additive* one, where each moving layer contributes some intensity to the final image (Szeliski, Avidan, and Anandan 2000).

If the motions of the individual layers are known, the recovery of the individual layers is a simple constrained least squares problem, with the individual layer images are constrained to be positive. However, this problem can suffer from extended low-frequency ambiguities, especially if either of the layers lacks dark (black) pixels or the motion is uni-directional. In their paper, Szeliski, Avidan, and Anandan (2000) show that the simultaneous estimation of the motions and layer values can be obtained by alternating between robustly computing the motion layers and then making conservative (upper- or lower-bound) estimates of the layer intensities. The final motion and layer estimates can then be polished using gradient descent on a joint constrained least squares formulation similar to (Baker, Szeliski, and Anandan 1998), where the *over* compositing operator is replaced with addition.

Figures 8.17 and 8.18 show the results of applying these techniques to two different picture frames with reflections. Notice how, in the second sequence, the amount of reflected light is quite low compared to the transmitted light (the picture of the girl) and yet the algorithm is still able to recover both layers.

Unfortunately, the simple parametric motion models used in (Szeliski, Avidan, and Anandan 2000) are only valid for planar reflectors and scenes with shallow depth. The extension of these techniques to curved reflectors and scenes with significant depth has also been studied (Swaminathan, Kang, Szeliski *et al.* 2002; Criminisi, Kang, Swaminathan *et al.* 2005), as has the extension to scenes with more complex 3D depth (Tsin, Kang, and Szeliski 2006).

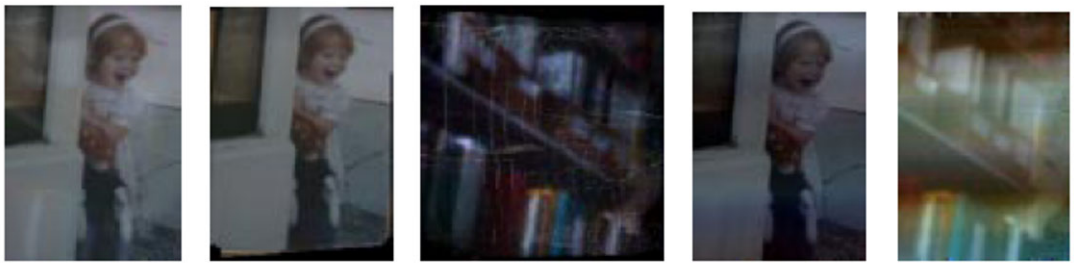


Figure 8.18 Transparent motion separation (Szeliski, Avidan, and Anandan 2000) © 2000 IEEE: (a) first image from input sequence; (b) dominant motion layer *min-composite*; (c) secondary motion residual layer *max-composite*; (d–e) final estimated picture and reflection layers. Note that the reflected layers in (c) and (e) are doubled in intensity to better show their structure.

8.6 Additional reading

Some of the earliest algorithms for motion estimation were developed for motion-compensated video coding (Netravali and Robbins 1979) and such techniques continue to be used in modern coding standards such as MPEG, H.263, and H.264 (Le Gall 1991; Richardson 2003).[14] In computer vision, this field was originally called *image sequence analysis* (Huang 1981). Some of the early seminal papers include the variational approaches developed by Horn and Schunck (1981) and Nagel and Enkelmann (1986), and the patch-based translational alignment technique developed by Lucas and Kanade (1981). Hierarchical (coarse-to-fine) versions of such algorithms were developed by Quam (1984), Anandan (1989), and Bergen, Anandan, Hanna *et al.* (1992), although they have also long been used in motion estimation for video coding.

Translational motion models were generalized to affine motion by Rehg and Witkin (1991), Fuh and Maragos (1991), and Bergen, Anandan, Hanna *et al.* (1992) and to quadric reference surfaces by Shashua and Toelg (1997) and Shashua and Wexler (2001)—see Baker and Matthews (2004) for a nice review. Such parametric motion estimation algorithms have found widespread application in video summarization (Teodosio and Bender 1993; Irani and Anandan 1998), video stabilization (Hansen, Anandan, Dana *et al.* 1994; Srinivasan, Chellappa, Veeraraghavan *et al.* 2005; Matsushita, Ofek, Ge *et al.* 2006), and video compression (Irani, Hsu, and Anandan 1995; Lee, ge Chen, lung Bruce Lin *et al.* 1997). Surveys of parametric image registration include those by Brown (1992), Zitov'aa and Flusser (2003), Goshtasby (2005), and Szeliski (2006a).

Good general surveys and comparisons of optic flow algorithms include those by Aggarwal and Nandhakumar (1988), Barron, Fleet, and Beauchemin (1994), Otte and Nagel (1994), Mitiche and Bouthemy (1996), Stiller and Konrad (1999), McCane, Novins, Crannitch *et al.* (2001), Szeliski (2006a), and Baker, Black, Lewis *et al.* (2007). The topic of matching primitives, i.e., pre-transforming images using filtering or other techniques before matching, is treated in a number of papers (Anandan 1989; Bergen, Anandan, Hanna *et al.* 1992; Scharstein 1994; Zabih and Woodfill 1994; Cox, Roy, and Hingorani 1995; Viola and

[14] http://www.itu.int/rec/T-REC-H.264.

Wells III 1997; Negahdaripour 1998; Kim, Kolmogorov, and Zabih 2003; Jia and Tang 2003; Papenberg, Bruhn, Brox *et al.* 2006; Seitz and Baker 2009). Hirschmüller and Scharstein (2009) compare a number of these approaches and report on their relative performance in scenes with exposure differences.

The publication of a new benchmark for evaluating optical flow algorithms (Baker, Black, Lewis *et al.* 2007) has led to rapid advances in the quality of estimation algorithms, to the point where new datasets may soon become necessary. According to their updated technical report (Baker, Scharstein, Lewis *et al.* 2009), most of the best performing algorithms use robust data and smoothness norms (often L_1 TV) and continuous variational optimization techniques, although some techniques use discrete optimization or segmentations (Papenberg, Bruhn, Brox *et al.* 2006; Trobin, Pock, Cremers *et al.* 2008; Xu, Chen, and Jia 2008; Lempitsky, Roth, and Rother. 2008; Werlberger, Trobin, Pock *et al.* 2009; Lei and Yang 2009; Wedel, Cremers, Pock *et al.* 2009).

8.7 Exercises

Ex 8.1: Correlation Implement and compare the performance of the following correlation algorithms:

- sum of squared differences (8.1)
- sum of robust differences (8.2)
- sum of absolute differences (8.3)
- bias–gain compensated squared differences (8.9)
- normalized cross-correlation (8.11)
- windowed versions of the above (8.22–8.23)
- Fourier-based implementations of the above measures (8.18–8.20)
- phase correlation (8.24)
- gradient cross-correlation (Argyriou and Vlachos 2003).

Compare a few of your algorithms on different motion sequences with different amounts of noise, exposure variation, occlusion, and frequency variations (e.g., high-frequency textures, such as sand or cloth, and low-frequency images, such as clouds or motion-blurred video). Some datasets with illumination variation and ground truth correspondences (horizontal motion) can be found at http://vision.middlebury.edu/stereo/data/ (the 2005 and 2006 datasets).

Some additional ideas, variants, and questions:

1. When do you think that phase correlation will outperform regular correlation or SSD? Can you show this experimentally or justify it analytically?
2. For the Fourier-based masked or windowed correlation and sum of squared differences, the results should be the same as the direct implementations. Note that you will have to expand (8.5) into a sum of pairwise correlations, just as in (8.22). (This is part of the exercise.)

3. For the bias–gain corrected variant of squared differences (8.9), you will also have to expand the terms to end up with a 3×3 (least squares) system of equations. If implementing the Fast Fourier Transform version, you will need to figure out how all of these entries can be evaluated in the Fourier domain.

4. (Optional) Implement some of the additional techniques studied by Hirschmüller and Scharstein (2009) and see if your results agree with theirs.

Ex 8.2: Affine registration Implement a coarse-to-fine direct method for affine and projective image alignment.

1. Does it help to use lower-order (simpler) models at coarser levels of the pyramid (Bergen, Anandan, Hanna *et al.* 1992)?

2. (Optional) Implement patch-based acceleration (Shum and Szeliski 2000; Baker and Matthews 2004).

3. See the Baker and Matthews (2004) survey for more comparisons and ideas.

Ex 8.3: Stabilization Write a program to *stabilize* an input video sequence. You should implement the following steps, as described in Section 8.2.1:

1. Compute the translation (and, optionally, rotation) between successive frames with robust outlier rejection.

2. Perform temporal high-pass filtering on the motion parameters to remove the low-frequency component (smooth the motion).

3. Compensate for the high-frequency motion, zooming in slightly (a user-specified amount) to avoid missing edge pixels.

4. (Optional) Do not zoom in, but instead borrow pixels from previous or subsequent frames to fill in.

5. (Optional) Compensate for images that are blurry because of fast motion by "stealing" higher frequencies from adjacent frames.

Ex 8.4: Optical flow Compute optical flow (spline-based or per-pixel) between two images, using one or more of the techniques described in this chapter.

1. Test your algorithms on the motion sequences available at http://vision.middlebury.edu/flow/ or http://people.csail.mit.edu/celiu/motionAnnotation/ and compare your results (visually) to those available on these Web sites. If you think your algorithm is competitive with the best, consider submitting it for formal evaluation.

2. Visualize the quality of your results by generating in-between images using frame interpolation (Exercise 8.5).

3. What can you say about the relative efficiency (speed) of your approach?

Ex 8.5: Automated morphing / frame interpolation Write a program to automatically morph between pairs of images. Implement the following steps, as sketched out in Section 8.5.1 and by Baker, Scharstein, Lewis *et al.* (2009):

1. Compute the flow both ways (previous exercise). Consider using a multi-frame ($n > 2$) technique to better deal with occluded regions.
2. For each intermediate (morphed) image, compute a set of flow vectors and which images should be used in the final composition.
3. Blend (cross-dissolve) the images and view with a sequence viewer.

Try this out on images of your friends and colleagues and see what kinds of morphs you get. Alternatively, take a video sequence and do a high-quality slow-motion effect. Compare your algorithm with simple cross-fading.

Ex 8.6: Motion-based user interaction Write a program to compute a low-resolution motion field in order to interactively control a simple application (Cutler and Turk 1998). For example:

1. Downsample each image using a pyramid and compute the optical flow (spline-based or pixel-based) from the previous frame.
2. Segment each training video sequence into different "actions" (e.g., hand moving inwards, moving up, no motion) and "learn" the velocity fields associated with each one. (You can simply find the mean and variance for each motion field or use something more sophisticated, such as a support vector machine (SVM).)
3. Write a recognizer that finds successive actions of approximately the right duration and hook it up to an interactive application (e.g., a sound generator or a computer game).
4. Ask your friends to test it out.

Ex 8.7: Video denoising Implement the algorithm sketched in Application 8.4.2. Your algorithm should contain the following steps:

1. Compute accurate per-pixel flow.
2. Determine which pixels in the reference image have good matches with other frames.
3. Either average all of the matched pixels or choose the sharpest image, if trying to compensate for blur. Don't forget to use regular single-frame denoising techniques as part of your solution, (see Section 3.4.4, Section 3.7.3, and Exercise 3.11).
4. Devise a fall-back strategy for areas where you don't think the flow estimates are accurate enough.

Ex 8.8: Motion segmentation Write a program to segment an image into separately moving regions or to reliably find motion boundaries.

Use the human-assisted motion segmentation database at http://people.csail.mit.edu/celiu/motionAnnotation/ as some of your test data.

Ex 8.9: Layered motion estimation Decompose into separate layers (Section 8.5) a video sequence of a scene taken with a moving camera:

1. Find the set of dominant (affine or planar perspective) motions, either by computing them in blocks or finding a robust estimate and then iteratively re-fitting outliers.
2. Determine which pixels go with each motion.
3. Construct the layers by blending pixels from different frames.
4. (Optional) Add per-pixel residual flows or depths.
5. (Optional) Refine your estimates using an iterative global optimization technique.
6. (Optional) Write an interactive renderer to generate in-between frames or view the scene from different viewpoints (Shade, Gortler, He *et al.* 1998).
7. (Optional) Construct an *unwrap mosaic* from a more complex scene and use this to do some video editing (Rav-Acha, Kohli, Fitzgibbon *et al.* 2008).

Ex 8.10: Transparent motion and reflection estimation Take a video sequence looking through a window (or picture frame) and see if you can remove the reflection in order to better see what is inside.

The steps are described in Section 8.5.2 and by Szeliski, Avidan, and Anandan (2000). Alternative approaches can be found in work by Shizawa and Mase (1991), Bergen, Burt, Hingorani *et al.* (1992), Darrell and Simoncelli (1993), Darrell and Pentland (1995), Irani, Rousso, and Peleg (1994), Black and Anandan (1996), and Ju, Black, and Jepson (1996).